

Data-efficient Robot Learning using Natural Human Guidance

by

Anthony Liang

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2026

Copyright 2026 Anthony Liang

Acknowledgements

This dissertation is the result of several years of work, much of which did not follow a clear or linear path. At times, the PhD felt difficult and there was no light at the end of the tunnel. I was often lost trying to find a research direction that was uniquely my own. Admittedly, I entered the program with a relatively narrow view of the field and without a clear plan for what I wanted to work on. I am grateful to Jesse Thomason for taking a chance on me and for his support and patience in those early years. Jesse set a high standard for academic writing, communication, and thinking which laid the foundations for the researcher I am today.

Early on, I spent a lot of time exploring, often going down one rabbit hole after another, some of which led to a dead end. I dabbled in a range of topics, from skill-based reinforcement learning to language models for dialogue systems, without a clear sense of what would stick. By my third year, when many students already had their first paper and a concrete roadmap for the rest of their PhD, I was still trying to find my footing and piece together a research narrative that felt coherent.

Around that time, Erdem Bıyık was joining USC as faculty after completing his postdoc. I had been getting interested in robot learning and ideating about how to teach robots more efficiently, and the timing worked out in a way that felt genuinely serendipitous. Erdem spent years working on closely related problems and him joining the department brought a breath of fresh air, helping me find clarity and focus in my research. I'm grateful for Erdem taking me under his wing and always being available to give feedback on my half-baked research ideas. I am also grateful to Stephen Tu, whom I first met while interning at Google. He is deeply thoughtful, inquisitive,

and grounded in theory, and working with him pushed me to be more skeptical and rigorous in how I approach problems. I also thank Gaurav Sukhatme for his pioneering efforts in the field, for treating everyone in the department like his own student, and for fostering a collaborative community. I also appreciate the Viterbi School of Engineering along with all the helpful people in the administration who helped along the way: Andy Chen, Lizsl De Leon, Ellecia Williams, and Asiroh Cham.

I want to thank my family, who have supported me unconditionally every step of the way. I am eternally grateful to my mom, Betty Lu, my dad, Kenny Liang, and my grandparents, who have made many many sacrifices so that I can pursue my selfish goals. Huge shoutout to my younger brother, Arthur Liang, who even at a young age has always aimed high. His ambition and drive have been a constant source of motivation for me. I'm also extremely grateful for my partner and best friend, Irene Huang, who has been a steady anchor for me through the lowest lows and highest highs of my PhD journey. I appreciate her for keeping me company through all the late nights spent working on paper deadlines and preparing for interviews. Thank you for reminding me that life is more than the next deadline and always giving me something to look forward to.

I'm also incredibly fortunate for the amazing group of friends I've grown close with at USC between GLAMOR, LiraLab, and the robotics department. Thank you to my first-ever labmates from GLAMOR who persevered with me through the first year woes: Abrar Anwar, Ishika Singh, Tejas Srinivasan, Ting-yun (Charlotte) Chang, and Leticia Pinto-Alva. Also, thank you to my current and former labmates from LiraLab for all the discussions and feedback that helped sharpen my research: Sumedh Sontakke, Jesse Zhang, Yutai Zhou, Yigit Korkmaz, Ayush Jain, Pavel Czempin, Ayano Hiranko, Yimin Tang, Bo-Ruei Huang, and Jiahui Zhang. I'm also grateful to have the opportunity to mentor some exceptional undergraduate and Master's students: Matthew Hong, Sankalp (Sunny) Agrawal, Jaiv Doshi, Yusen Luo, Cagan Bakirci, Ziyi Liu, Shreya Ramanujam, and Mihir Rao. I additionally thank my peers and friends who have helped me grow as a researcher and an individual: Ya-chuan (Sophie) Hsu, Wang (Bill) Zhu, Zhaojing Yang, Jeremy Morgan, Min-

june Hwang, Lee Kezar, and many others. Also thank you to my amazing friends who have been a part of this journey from the start: Frankie Li, Leon Cheng, Jessy Ma, Carson Fiechtner, Olivia Guan, Isabel Chen, Shaeq Ahmed, Jennifer Chou, and all the other friends I've made along the way.

I'm also grateful for all my collaborators outside of academia where I was fortunate to have interned at Google Research with Chih-wei Hsu, Yinlam Chow, Guy Tennenholtz and Craig Boutilier, Meta Reality labs with Paul Crook, Andrea Madotto, and Stephen Roller, and Google DeepMind with Kalpesh Krishna, Abhimanyu Goyal, Jacob Eisenstein, Adam Fisch, and Jonathan Berant. These internship experiences have helped me polish my skills as a researcher and engineer and given me newfound perspectives on how my research could have tangible impact on the broader community.

I want to especially thank everyone who I spent quality time with outside of a research setting where I was able to destress and forget about work: Jesse, Yutai, and Sophie for being my regular gym buddies, Sumeet, Sumedh, and Tejas for entertaining me in tennis, Ayush, Sophie and Ishika for badminton sessions at LABC, Sid, Jesse, and Abrar, for Valorant gaming nights, and Sid, Grace, and Ishika for the yearly ski adventures.

Contents

Acknowledgements	ii
List of Tables	xi
List of Figures	xvii
Abstract	xxv
Chapter 1: Introduction	2
1.1 Scalable Supervision from Human Signals	4
1.2 Autonomous Improvement under Distribution Shift	5
Chapter 2: Background: Reinforcement Learning, Imitation Learning, and Offline RL	7
2.1 Markov decision processes and reinforcement learning	7
2.2 Imitation learning	8
2.3 Offline reinforcement learning	9
I Scalable Supervision from Human Signals	10
Part I: Scalable Supervision from Human Signals	11
Chapter 3: ViSaRL: Visual Reinforcement Learning Guided by Human Saliency	14
3.1 Introduction	14
3.2 Related Work	16
3.3 Visual Saliency-Guided Reinforcement Learning	18
3.4 Experiment Setup	20
3.5 Simulation Experiments	22
3.5.1 CNN Encoder	23
3.5.2 MultiMAE Transformer	24
3.6 Real Robot Experiments	25
3.7 Conclusion	27
Chapter 4: HAND Me the Data: Fast Robot Adaptation via Hand Path Retrieval	29
4.1 Introduction	29

4.2	Related Works	31
4.3	HAND: Fast Robot Adaptation via Hand Path Retrieval	32
4.3.1	Preliminaries and Formulation	32
4.3.2	Path Distance as a Unifying Representation for Retrieval	34
4.3.3	Retrieving Relevant Sub-Trajectories using Path Distance	35
4.3.4	Putting it All Together: Fast-Adaptation with Parameter-Efficient Policy Fine-tuning	36
4.4	Experiments	38
4.4.1	Experimental Setup	38
4.4.2	Experimental Evaluation	40
4.5	Conclusion and Limitations	45
Chapter 5:	CLAM: Continuous Latent Action Models for Robot Learning from Unla- beled Demonstrations	47
5.1	Introduction	47
5.2	Related Work	49
5.3	Problem Setting	50
5.4	Continuous Latent Action Models	51
5.4.1	Latent Action Model Training	52
5.4.2	Latent Action Policy Training	54
5.5	Experimental Setup	54
5.6	Results	57
5.6.1	CLAM Design Choices	62
5.7	Conclusion	63
5.8	Limitations	64
Chapter 6:	Inference-Time Plan Guidance with Interleaved Reasoning	66
6.1	Introduction	66
6.2	Interleaved Reasoning	68
6.3	Inference-Time Scaling	71
6.4	Experimental Setup	73
6.5	Results	76
6.6	Related Work	81
6.7	Conclusion	83
II	Autonomous Improvement under Distribution Shift	85
Part II:	Autonomous Improvement under Distribution Shift	86

Chapter 7:	DynaMITE-RL: A Dynamic Model for Improved Temporal Meta-Reinforcement Learning	89
7.1	Introduction	89
7.2	Background	91
7.3	Dynamic Latent Contextual MDPs	93
7.4	DynaMITE-RL	95
7.5	Experiments	99
7.6	Related Work	105
7.7	Conclusion	106
Chapter 8:	Robometer: Scaling General-Purpose Robotic Reward Models via Trajectory Comparisons	108
8.1	Introduction	108
8.2	Related Works	111
8.3	Robometer	113
8.3.1	RBM-1M Dataset	113
8.3.2	Robometer Architecture and Tokenization	114
8.3.3	Training Objectives	115
8.3.4	Data Sampling and Augmentation	116
8.4	Experiments	118
8.4.1	Reward Evaluation	119
8.4.2	Ablations: Why does Robometer Perform so Well?	122
8.4.3	Accelerating Robot Learning with Generalizable Rewards	125
8.5	Limitations and Future Work	129
Chapter 9:	Concluding Remarks and Future Directions	132
	Bibliography	136
Appendix A:	ViSaRL	167
A.1	Environment and Task Details	167
A.2	Hyperparameters	168
A.3	Implementation Details	169
A.4	Real Robot Imitation Learning	169
A.5	Saliency Predictor	171
Appendix B:	HAND	176
B.1	Environment Details and Hyperparameters	176

B.1.1	CALVIN.	176
B.1.2	Real Robot Experimental Setup	177
B.2	HAND Algorithm	181
B.3	User Studies	182
B.3.1	Efficiency of Hand Demonstrations	182
B.3.2	Fast Adaptation to Long-Horizon Tasks	182
B.4	Qualitative Retrieval Analysis	184
B.5	CALVIN Results	186
B.6	Real Robot Results	188
Appendix C:	CLAM	189
C.1	Additional Experimental Results	189
C.1.1	State-based Results MetaWorld	189
C.1.2	CLAM Ablations	190
C.1.3	Additional Transfer Experiments in CALVIN	191
C.2	Trivial Solutions	192
C.3	Detailed Algorithms	193
C.4	Hyperparameters	194
C.5	Environment Details	197
C.6	Real Robot Experiments	199
C.7	Architecture Details	202
C.7.1	Behavior Cloning Policy	202
C.7.2	Space Time CLAM	204
C.7.3	Latent Action Model.	205
C.8	Data Generation	211
C.9	Baseline Details	211
Appendix D:	Inference-Time Plan Guidance	214
D.1	Training and inference algorithms	215
D.1.1	PLANTAIN Algorithm	215
D.2	Prompts, datasets, and implementation	217
D.2.1	Interleaved Reasoning System Instruction	217
D.2.2	LLM Plan Autorater	218
D.2.3	Dataset Details	219
D.2.4	Training and Evaluation Datasets	219
D.2.5	Training Details	221

D.2.6	Synthetic Interleaved Response Generation	222
D.3	Additional results and examples	223
D.3.1	Training dataset prompt examples	223
D.3.2	Generated Plan and Answers	227
D.3.3	MATH500 Think-Answer vs. Rewind-and-Repeat	229
Appendix E:	DynaMITE-RL	233
E.1	ELBO derivation for DLCMDP	233
E.2	Pseudocode for DynaMITE-RL	235
E.3	Additional experimental results	236
E.4	Evaluation environments	237
E.5	Implementation details and hyperparameters	241
E.5.1	Compute Resources and Runtime	244
Appendix F:	Robometer	245
F.1	Dataset Details	245
F.1.1	Individual RBM-1M Training Dataset Details	248
F.1.2	Dataset Filtering and Task End-State Adjustment	254
F.1.3	Individual Evaluation Dataset Details	255
F.2	Model Details	256
F.2.1	Model Architecture and Training Parameters	256
F.2.2	Training Objectives	258
F.2.3	Computational Resources	262
F.3	Additional Reward Evaluation Results	263
F.3.1	Preference Prediction	265
F.3.2	Robometer in Long Horizon Tasks	268
F.3.3	RoboRewardBench Evaluation	268
F.4	Additional Ablations	270
F.4.1	Preference pair sampling ablations	270
F.4.2	RBM-EVAL-00D Pre-Training Objective Ablations	271
F.4.3	Training loss weight ablations	272
F.5	Policy Learning Experiment Details	272
F.5.1	RL with Ablated Reward Models	272
F.5.2	Automatic Online RL	272
F.5.3	Failure Detection	276
F.5.4	Data Filtering & Retrieval	281
F.5.5	Combining Noisy and Expert Trajectories via Offline RL	283

F.6	Low-rank Fine-tuning Robometer-4B	285
F.7	Robometer with Model-Based RL	287

List of Tables

3.1	Success rate on four Meta-World manipulation tasks averaged across 50 rollouts and 3 seeds for the CNN and MultiMAE visual encoder backbones. Text in maroon indicates the best performing method per task.	22
3.2	Average return on DMC-GB tasks for the <code>color</code> and <code>video</code> evaluation settings.	23
3.4	Human-annotated saliency versus depth and predicted surface normals as additional MultiMAE inputs. <i>SN</i> denotes <i>surface normal</i>	25
3.3	Adding saliency improves performance across tasks and modalities for MultiMAE. <i>SN</i> denotes <i>surface normal</i> (columns match Table 3.4).	25
3.5	RGB vs. RGB+Saliency (ours) with absolute improvements in task success.	26
3.6	Human-eval successes (trials out of 10) with saliency improvements on real-robot tasks.	26
3.7	Task success rates in real-world tabletop manipulation tasks for RGB and RGB+Saliency with MultiMAE.	27
4.1	HAND retrieves more sub-trajectories performing the task.	40
4.2	Camera angle robustness results. Number of relevant retrieved trajectories for PUT LID ON POT if we change the camera angle vertically and horizontally by 10° increments. HAND retrieves +18% more relevant trajectories compared to STRAP even in the extreme case of 30° shift.	41
4.3	Long horizon COOK CARROT task results. We show success rates on each subtask and on the full task execution. Successes are out of 10.	42
4.4	Hand vs. Robot Teleoperation. HAND uses a single hand demonstration while STRAP uses robot teleoperation demonstrations. HAND achieves higher success rates in significantly less data collection time.	44
5.1	MetaWorld State-Based Results. We report normalized returns for DMControl tasks and average task success rates for MetaWorld tasks. Maroon highlights the best method in each environment, excluding BC-Expert , which is trained with expert labeled data. Our methods, denoted by an asterisk (*), outperform all baselines across tasks.	55

5.2	CLAM Ablation Study. Both continuous latent actions and joint training are necessary to improve task success. Using continuous actions (row 2 → row 3) yields about a 1.5× improvement in success rates. Joint training (row 3 → row 4) further boosts performance by roughly 3×.	60
5.3	Real Robot Results. ST-CLAM significantly outperforms baseline methods across all tasks. We report task success rate out of 10 trials; partial success is provided for completing a subtask.	61
6.1	Interleaved reasoning generalizes beyond coding tasks, reducing latency and preserving accuracy. Plan-first decoding (<i>Plan-Answer</i>) substantially lowers time-to-first-response (TTFR) while maintaining or improving pass@1 accuracy compared to the standard <i>Think-Answer</i> baseline with a token budget of 4096. Despite being post-trained only on coding data, the learned interleaved behavior transfers effectively to math, text-to-SQL, and reading comprehension tasks. . . .	77
6.2	QWEN3-8B at an 8192-token decoding budget. Pass@1 accuracy (%) on MATH500, MBPP, Text-to-SQL, and QuaLITY. We compare Think-Answer (TA) and an RL-only interleaved baseline [305] against our supervised-then-RL Plan-Answer model with inference-time strategies Best-of-N (5) and R&R (Plan). Best per column in bold	77
6.3	Inference-time strategies: Plan-level feedback provides earlier and more informative supervision than answer-level feedback, yielding higher accuracy and lower TTFR. Both Best-of-N and R&R (Plan) improve over Plan-Answer without additional training.	78
6.4	Token-budget ablation: Interleaved inference achieves near-saturated accuracy at 4K tokens, highlighting efficient use of available context. While Think-Answer continues to improve with longer reasoning—reflecting redundant token usage—Plan-Answer benefits modestly from larger budgets but remains below Rewind-and-Repeat, indicating stable yet non-adaptive scaling. Rewind-and-Repeat achieves the strongest token efficiency and overall accuracy.	79
6.5	Tokens per problem (T/P, ↓) at an 8192-token context window. We report total generated tokens per evaluation instance for Think-Answer (TA), the RL-only interleaved baseline of Xie et al. [305], and our interleaved inference strategies. Lower is better; best per column within each model in bold	80
6.6	Rewind statistics. Fraction of prompts requiring plan rewinds and corresponding approval rates. On average, only 22% of plans require a rewind, and over 80% of corrections succeed within two attempts, demonstrating the efficiency of plan-level R&R.	81
7.1	Average single episode returns for DynaMITE-RL and other state-of-the-art meta-RL algorithms across different environments. Results for all environments are averaged across 5 seeds beside ScratchItch which has 3 seeds. DynaMITE-RL , in bold, achieves the highest return on all of the evaluation environments and is the only method able to recover an optimal policy.	102

7.2	Average single episode returns with offline RL. Results are averaged across 5 random seeds. Algorithm with the highest average return are shown in bold. We present results for an oracle agent trained with goal information for reference.	103
8.1	(a) Reward alignment (VOC Pearson r) and (b) trajectory ranking (Kendall τ_a) on RBM-EVAL datasets. Baselines are split into categories based on training data: GVL and TOPReward training data is unknown, VLAC is trained on a 300k-trajectory dataset, and RoboDopamine is trained on a 100k-trajectory dataset. We compare Robometer against RoboReward-4B/8B with their own training data, and we also evaluate ReWiND and Robometer trained with the full RBM-1M dataset. Kendall τ_a is not calculated for RBM-EVAL-ID due to it only having simulation failure data.	118
8.2	Finetuning Robometer on RoboFAC dataset. Zero-shot performance is strong, and fine-tuning the base Qwen3 VLM performs worse on Kendall τ and success - fail difference compared to zero-shot Robometer. Meanwhile, Robometer fine-tuned performs best, either with LoRA or FFT.	122
8.3	Reward alignment (VOC Pearson r), policy ranking (Kendall τ), and average reward difference between successful and failed trajectories on LIBERO-90 and RBM-EVAL-00D.	123
8.4	Failure detection performance. Robometer achieves the highest average F1 score. T.U. is the token-uncertainty baseline.	128
A.1	Hyperparameters used in SAC training.	169
A.2	Hyperparameters used in the real robot experiment.	170
A.3	Evaluation metrics measuring the prediction quality different state-of-the-art predictors.	173
A.4	Effect of training dataset size on saliency prediction	173
B.1	CALVIN Close Drawer: Performance with and without expert demonstrations	186
B.2	CALVIN Move Slider Left: Performance with and without expert demonstrations	187
B.3	CALVIN Turn On LED: Performance with and without expert demonstrations	187
B.4	Real-world expert demonstrations ($N = 3$). Success rates out of 10 trials per task.	188
B.5	Real-world hand demonstration ($N = 1$). Success rates out of 10 trials per task.	188
C.1	State-Based Input Results (50 action-labeled trajectories)	190
C.2	State-Based Input Results (100 action-labeled trajectories). We report average task success rate for MetaWorld tasks. Maroon highlights the best method in that environment except for BC-Expert which is trained with expert, labeled data. Our method TF-CLAM, which uses a transformer IDM/FDM, outperforms the baselines on all tasks. *Methods with an asterisk are ours.	190

C.3	CLAM ablation study (state-based). We ablate discrete vs. continuous actions and with/without joint training. Both are critical for a performant policy; joint training improves results across tasks.	191
C.4	Varying action decoder loss weight β . Increasing the decoder weight improves downstream task performance.	191
C.5	CALVIN generalization results. CLAM generalizes better than VPT on target tasks without labeled demonstrations for those tasks.	191
C.6	Image reconstruction loss vs. cheating loss	193
C.7	BC Hyperparameters	195
C.8	CLAM Pretraining Hyperparameters	195
C.9	Transformer CLAM Model Hyperparameters	196
C.10	CALVIN Transformer CLAM Model Hyperparameters	196
C.11	MetaWorld Environment Hyperparameters	196
C.12	CALVIN Environment Hyperparameters	197
C.13	Data splits (# trajectories).	197
C.14	Task observation and action dimensions.	199
C.15	Environment comparison summary.	199
D.1	SFT dataset details.	219
D.2	RL dataset details.	220
D.3	Training hyperparameters.	221
E.1	Ablation study over different values of β in the HalfCheetah-Vel environment. . .	237
E.2	Training hyperparameters. Dashed entries means the same value is used across all environments.	242
E.3	Hyperparameters for Transformer Encoder	242
F.1	Comparison of known approximate trajectory counts across recent general-purpose reward modeling papers.	245
F.2	RBM-EVAL-ID - In distribution evaluation datasets overview. # Trajs. is computed by aggregating all evaluation splits per dataset, counting unique annotated trajectories.	246
F.3	RBM-EVAL-OOD - Out-of-distribution evaluation datasets overview. # Trajs. is computed by aggregating all evaluation splits per dataset, counting unique annotated trajectories.	246
F.4	RBM-1M Training datasets overview. # Trajs. is determined by counting every unique video-language annotation (possibly across multiple views when available).	247
F.5	Training Configuration for Robometer	256

F.6	Dataset-specific success cutoffs used to correct for delayed episode termination in teleoperated data. Datasets not contained here use a default success cutoff of 1.0.	260
F.7	Bradley-Terry (BT) versus preference label from a dedicated $\langle \text{pref_token} \rangle$ (BCE) given both videos in a single forward pass, evaluated on RBM-EVAL-OOD.	262
F.8	RL-VLM-F vs Robometer on Trajectory Quality. Different quality trajectory pairwise preference accuracy on 500 comparisons for each of the RBM-EVAL-OOD datasets.	266
F.9	RL-VLM-F vs Robometer on Different Tasks. Different task trajectory pairwise preference accuracy on 500 comparisons for each of the RBM-EVAL-OOD datasets.	267
F.10	Per-dataset model ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failed trajectories on RBM-EVAL-OOD	267
F.11	Per-dataset VOC reward alignment results. Pearson correlation results (VOC) across individual datasets for RBM-EVAL-ID and RBM-EVAL-OOD.	267
F.12	Per-dataset trajectory ranking results. Trajectory ranking results on individual RBM-EVAL-OOD datasets.	268
F.13	Evaluation on the RoboRewardBench benchmark [175].	269
F.14	Data sampling strategy ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failure trajectories on RBM-EVAL-OOD and LIBERO-90. Models are trained without using RBM-EVAL-OOD scenes; RBM-EVAL-OOD and LIBERO-90 are held out for evaluation.	270
F.15	Loss weight ablation results. Hyperparameter sweep over preference/progress loss weights ($\lambda_{\text{pref}}, \lambda_{\text{prog}}$) on LIBERO-10 and LIBERO-90. Each model is trained on LIBERO-(10, Spatial, Object, Goal) and LIBERO-90 is heldout for evaluation only.	270
F.16	Success prediction head ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failure trajectories on RBM-EVAL-OOD and LIBERO-90.	271
F.17	RL with Ablated Reward Models. SAC hyperparameters.	273
F.18	Automatic Online RL. DSRL SAC hyperparameters.	274
F.19	Automatic Online RL reward relabeling. True positives (TP) and false positives (FP) for reward predictions during online RL.	276
F.20	Failure detection performance. We report true positive rate (TPR; correctly detecting failures), true negative rate (TNR; correctly identifying successful executions), and F1 score.	278
F.21	Trajectory retrieval analysis. Top-5 retrieval quality summary per task. Counts indicate how many of the top-5 retrieved trajectories are labeled as success (Succ), suboptimal (Subopt), failure (Fail), or unrelated (Unrel).	283
F.22	SO-101 offline RL experiment. Hyperparameters used for Implicit Q-Learning (IQL) in offline policy training.	285

F.23	SO-101 offline RL model architectures.	Transformer-based actor and critic architectures used for offline IQL training. Both networks share a lightweight Transformer encoder, followed by task-specific MLP heads.	286
------	---	---	-----

List of Figures

3.1	ViSaRL trains a saliency prediction model from a few human-annotated saliency maps. This model is used to augment an offline image dataset with saliency. A visual encoder is pretrained with the dataset and used during downstream policy learning to generate latent representations of the agent’s observations.	15
3.2	ViSaRL. We pretrain a MultiMAE [12] Transformer on a dataset of paired images and saliency maps. MultiMAE employs a self-supervised objective in which masked patches for both input modalities are reconstructed given only the visible patches. The pretrained model is frozen and used for extracting representations during task learning. There is no input masking during downstream RL.	18
3.3	Learning curves for four robot manipulation tasks in Meta-World evaluated by task success rate. (Top) CNN encoder methods. (Bottom) Transformer encoder methods. We select tasks that require manipulating small objects with different motions such as a pushing, pulling, and reaching. The solid lines represent the mean and shaded region the standard error across three seeds.	21
3.4	Annotation interface. Click-based saliency collection (Gaussians at clicked pixels). Warmer colors denote more salient regions such as the drawer handle and the robot end-effector.	21
3.5	Evaluation Tasks. Four Meta-World (top) simulation tasks and four real-robot tabletop manipulation tasks (bottom).	26
4.1	HAND learns a policy from as little as one (1) human hand demonstration.	30
4.2	HAND enables fast-adaptation to a new target task by using an easy-to-provide hand demonstration of the target task (Left). We propose a two-step retrieval procedure where we first filter the trajectories in the offline play dataset, $\mathcal{D}_{\text{play}}$, for visually similar trajectories based on features from a pretrained vision model. We use off-the-shelf, pretrained hand detection and point tracking to construct 2D paths of the motion for both the human hand and robot end-effector. We use these paths as a distance metric to retrieve relevant trajectories from the play dataset (Middle) for quickly fine-tuning a pretrained transformer policy on the target task (Right).	33
4.3	WidowX Robot Arm Setup. We evaluate the scalability of HAND on 10 manipulation tasks on a WidowX robot arm in a kitchen setup [292].	38

4.4	Qualitative retrieval results on OOD scene. We visualize the top sub-trajectory match of Flow , STRAP , HAND(-VF) , and HAND on two OOD domain demonstrations recorded from an iPhone camera, showing approaching a K-Cup and putting it into the machine. Only HAND 's top match is relevant for both hand demonstrations.	41
4.5	Fast Adaptation Study. We conduct a small-scale user study to demonstrate HAND 's ability to learn robot policies in real-time. From providing the hand demonstration (Left), to retrieval and fine-tuning a base policy (Middle), to evaluating the policy (Right), we show HAND can learn to put a carrot in the blender with 7.5/10 task completion in less than 4 minutes.	43
4.6	Real Robot Results. Task completion (including partial completion) out of 10 of π_{base} , STRAP , Flow , and HAND	43
5.1	Overview of CLAM. CLAM consists of a latent inverse dynamics model , f_ϕ , which infers the latent action between consecutive observations and latent forward dynamics model , g_ψ , which predicts the future observation conditioned on the latent action and observation history. CLAM learns a latent action space through the self-supervised objective of future observation reconstruction. Unlike prior work, CLAM produces continuous latent actions . To ensure the learned latent space is amenable to decoding to real-world actions, CLAM jointly trains the action decoder and the latent action model.	48
5.2	MetaWorld Image-Based Experiments. Task success rates are averaged over 50 evaluation rollouts across 3 random seeds using image-based inputs. All methods are trained with the same amount of action-labeled data (100 trajectories) for fair comparison. Because D_{labeled} is not necessarily expert data, baselines struggle to learn a performant downstream policy, whereas MLP-CLAM and Trans.-CLAM achieve up to a 3 $\times$ improvement over the best baseline. Baselines using a <i>discrete</i> latent action space are indicated with hashed markers. For reference, we also report results for BC-Expert , trained on the same number of <i>expert</i> trajectories to illustrate the ideal performance BC can achieve with ground-truth action labels. .	56
5.3	WidowX Robot Arm Setup and Evaluation Tasks. <i>Left:</i> WidowX robot arm in a toy kitchen setup [292]. We use an Intel Realsense D435 RGBD camera as a fixed external camera and a Logitech C920 webcam as an over-the-shoulder camera view. <i>Right:</i> Four tabletop manipulation tasks used to evaluate the scalability of CLAM.	58
5.4	(Left) Increasing the latent action dimension improves the model expressivity for policy learning. Up until a latent dimension of 4, the learned latent action space fails to be useful for imitation learning. However, a latent dimension of 8 has sufficient capacity, achieving 57% success rate on the Assembly task. (Right) CLAM scales with the amount of unlabeled video data. The performance of the downstream policy improves as we annotate more trajectories using the pretrained CLAM.	61

5.5	(Left) We vary the number of labeled trajectories for training the action decoder. While BC performance struggles to learn from <i>non-expert data</i> , our method improves with more data. (Right) We also evaluate the robustness of CLAM to varying expertise of data. We learn a better policy than BC with the same amount of labeled random trajectories. Unsurprisingly, with expert data, our method recovers an optimal policy.	62
6.1	PLANTAIN Rewind-and-Repeat After post-training an interleaved reasoning model, we apply an iterative rejection-sampling strategy for inference-time plan guidance. PLANTAIN first produces a <i>plan</i> as its initial intermediate response. This plan is evaluated by a human judge or an LLM autorater. If accepted, the model proceeds to generate the final answer. If rejected, the process is <i>rewound</i> such that the rejected plan is appended to a history of failures, and the model is re-prompted to produce a new, distinct plan. Crucially, the response generated by PLANTAIN is typically of higher quality than the response produced by a standard Think-Answer model. Additionally, subsequent thinking blocks after a rejection plan are short because the model only needs to adjust the plan rather than regenerate full reasoning. This leads to early pruning of suboptimal reasoning paths and closer alignment with true user intent.	67
7.1	(Left) The graphical model for a DLCMDP. The transition dynamics of the environment follows $T(s_{t+1}, m_{t+1} \mid s_t, a_t, m_t)$. At every timestep t , an i.i.d. Bernoulli random variable, d_t , denotes the change in the latent context, m_t . Blue shaded variables are observed and white shaded variables are latent. (Right) A DLCMDP rollout. Each session i is governed by a latent variable m^i which is changing between sessions according to a fixed transition function, $T_m(m' \mid m)$. We denote l_i as the length of session i . The state-action pair (s_t^i, a_t^i) at timestep t in session i is summarized into a single observed variable, x_t^i . We emphasize that session terminations are not explicitly observed.	91
7.2	VariBAD does not model the latent context dynamics and fails to adapt to the changing goal location. By contrast, DynaMITE-RL correctly infers the transition and consistently reaches the rewarding cell (green cross).	92
7.3	Pseudo-code (online RL training) and model architecture of DynaMITE-RL.	95
7.4	The environments considered in evaluating DynaMITE-RL. Each environment exhibits some change in reward and/or dynamics between sessions including changing goal locations (left and middle left), changing target velocities (middle right), and evolving user preferences of itch location (right).	97
7.5	Learning curves for DynaMITE-RL and state-of-the-art baseline methods. Shaded areas represent standard deviation over 5 different random seeds for each method and 3 for ScratchItch. In each of the evaluation environments, we observe that DynaMITE-RL exhibits better sample efficiency and converges to a policy with better environment returns than the baseline methods.	100

7.6	Ablating individual components of DynaMITE-RL . We observe that modelling latent dynamics is crucial in achieving good performance in a DLCMDP. Additionally, both consistency regularization and session reconstruction are critical for improving the sample efficiency and convergence to a better performing policy.	102
7.7	Ablation studies on various frequencies of latent context switches within an episode in the HalfCheetah-Vel environment. The boxplot shows the distribution over evaluation returns for 25 rollouts of trained policies with VariBAD and DynaMITE-RL . When $p = 0$, we have a latent MDP and when $p = 1$ this is equivalent to a general POMDP.	103
8.1	Robometer overview. Robometer is trained on RBM-1M, a 1M-trajectory dataset spanning 21 robot embodiments, containing both reward-labeled expert trajectories and reward-unlabeled failed trajectories. The model is supervised with a dual objective: predicting frame-level task progress (reward) and learning trajectory-level preferences from pairwise comparisons. It is also trained to predict per-frame task success for downstream RL.	108
8.2	Robometer is a VLM-based reward model that predicts dense, per-frame progress-based rewards and success labels for the first of two video trajectories. To scalably train with failure data, we also predict which of the two video trajectories better completes the task (preference). We use three strategies for curating training examples from our given datasets, which are further detailed in Section 8.3.4 with model architecture shown in Appendix Figure F.2. At inference, the model can input up to two trajectories: with a single trajectory, it outputs per-frame progress & success predictions; with two, it additionally predicts a preference for which trajectory best performs the task.	113
8.3	Video-Language Reward Confusion Matrix. For each task sampled at random from <i>self-collected</i> , <i>unseen</i> data from RBM-EVAL-00D, we compute rewards for all combinations of demonstration videos and language descriptions. Robometer produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions. We also report the column-normalized diagonal mean under each model, which represents the fraction of the model’s total reward for aligned task and video pairs.	116
8.4	Qualitative Analysis of Failure, Suboptimal and Successful Trajectories. We visualize the progress predictions for three trajectories of different quality for the same task. Notably, for the suboptimal trajectory, Robometer predicts steadily increasing progress as the robot approaches the pen holder, but sharply reduces its progress estimate when the marker is dropped, correctly reflecting regression in task completion. In contrast, RoboReward continues to assign high progress despite the task failure. Finally, Robometer is the only model that correctly predicts task success for the successful trajectory (i.e., high final progress value and explicit success prediction).	120

8.5	RL w/ Ablation Models in LIBERO-90 tasks from scratch, corresponding to ablations trained only on LIBERO-10/Object/Goal/Spatial data from Table 8.3. We report the average success rate $\pm$ standard deviation across 5 seeds.	123
8.6	Automatic online RL with DSRL [291] on a DROID [149] setup with Robometer improves π_0 from 20% to 85% on a single-stage task and 20% to 70% on a two-stage task, outperforming RoboReward ’s overall success rate by 2.5 $\times$. DSRL with Robometer learns to avoid base π_0 errors such as collisions or moving the wrong object. The setup is deemed “automatic” because success detection and stage advancement are handled automatically by the reward model, requiring human intervention only for physical scene resets.	125
8.7	Offline RL results using IQL on a mixture of Noisy and Expert trajectories. Robometer rewards consistently outperform both RoboReward and sparse rewards: 2.4 $\times$ average success rate improvement over the best baseline for each task.	126
8.8	(a): Proportion of task-relevant subtrajectories out of 100 retrieval queries. Our method consistently retrieves a high number of relevant subtrajectories using either the preference or progress objective. (b): Success rates of LoRA-finetuned $\pi_{0.5}$ policies using the retrieved trajectories from each method. Small amounts of suboptimal & unrelated data retrieved by other baselines degrade policy-learning performance: Robometer -retrieval attains an average 4.5 $\times$ success rate improvement over the best baseline.	128
A.1	DMControl Generalization Benchmark (Top) Four continuous control tasks from the DMControl suite. (Middle and Bottom) We evaluate on the color_easy and video_easy settings to test the generalizability of the learned representations.	168
A.2	Robot Policy Architecture We use an LSTM policy with MLPs to embed the input and generate continuous actions. The input is the cropped RGB image from an external camera and the predicted saliency map. The visual input is concatenated with the proprioceptive information.	170
A.3	MultiMAE predictions for different random masks. We visualize the masked predictions for RGB observation from each of the four tasks. For each input image, we randomly sample three different masks from a uniform distribution between RGB and saliency. Only 1/4 of the total patches are unmasked. Even when there are a few unmasked patches from one modality, the reconstructions are still very accurate due to cross-modal interaction. Saliency maps are shown with color for the purposes of visualization.	171
A.4	Zero-shot inference with pretrained PiCANet model for Meta-World observations. We observe that using only a pretrained saliency predictor will only identify the most prominent entity in the scene which in our case is the robot arm. However, task-relevant objects such as the block or the drawer, which are typically smaller in the image frame relative to the robot arm, are not highlighted.	172

A.5	Visualization of randomly sampled environment observations, their corresponding saliency predicted by a pretrained saliency model, and masked RGB observation from the MetaWorld tasks.	174
A.6	Visualization of randomly sampled environment observations, their corresponding saliency predicted by a pretrained saliency model, and masked RGB observation from real robot task demonstrations.	175
B.1	Environments. We retrieve data from a prior dataset to train on <i>new scenes</i> in CALVIN. On our real-world WidowX-250 robot, we demonstrate real-world learning from Robometer-retrieved trajectories along with real-time adaptation to long-horizon tasks.	176
B.2	Real Robot Tasks. We evaluate HAND on 5 different real robot tasks. The last two are long-horizon tasks, requiring more than 100 timesteps of execution. . . .	177
B.3	Real Robot Policy Architecture. (Left) Learnable image embeddings following [295]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [338]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.	179
B.4	Task Rollouts	180
B.5	Efficient Demonstrations. Two users, unfamiliar with HAND are asked to collect trajectories either via teleoperation (Left) or using their hands (Right). HAND retrieval achieves a 50% success rate with the same amount of demonstrations using 3× less time. STRAP retrieval is unable to reach 50% even when provided with more expert demonstrations.	182
B.6	Fast Adaptation. We conduct a small-scale user study to demonstrate HAND’s ability to learn robot policies in real-time. From providing the hand demonstration (Left), to retrieval and fine-tuning a base policy (Middle), to evaluating the policy (Right), we show that HAND can learn to solve the Blend Carrot task with over 70% success rate in less than 3 minutes.	183
B.7	Qualitative Retrieval Examples. We show the top 5 matches from $\mathcal{D}_{\text{play}}$ for STRAP (top) and HAND (bottom) provided the hand demonstration for each of our evaluation tasks.	185
C.1	Evaluation environments in simulation. We evaluate our approach on both <i>locomotion</i> tasks from the DMControl benchmark (Hopper and HalfCheetah) and <i>manipulation</i> tasks (Assembly, Bin Picking, Peg Insert Side, and Shelf Place) from the MetaWorld benchmark. We also evaluate in CALVIN with the Close Drawer and Slider Left tasks.	198
C.2	Real Robot BC Policy. (Left) Learnable image embeddings following [295]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [338]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.	201

C.3	Real Robot Evaluation Rollouts. We evaluate CLAM on four manipulation tasks. Each row is an example evaluation rollout and we subsample representative frames interpolated between the first and last timestep.	202
C.4	Space-Time Attention Encoder Block Model architecture for the Space-Time Attention Encoder Block.	204
C.5	Space-Time Attention Decoder Block Model architecture for the Space-Time Attention Decoder Block. Additional multihead attention to condition the learned representation on the latent action outputs from the IDM.	204
C.6	Space-Time CLAM. Model architecture for the Space-Time Latent Action Model.	205
C.7	Task Success Rate for Single-Task TD-MPC2 Agents. We train single-task agents using TD-MPC2 [113] to generate an offline dataset with different levels of behavior. All four agents exhibit expert performance on their respective tasks by the end of training.	212
E.1	Average test-time performance on MuJoCo tasks and ScratchItch task, trained separately with 5 seeds for MuJoCo tasks and 3 for itching task. The meta-trained policies are rolled out for 5 episodes to show how they adapt to the task. The returns averaged across the task with 95% confidence intervals shaded. We demonstrate that in our DLCMDP setting, the baseline methods struggle to adapt to the changing dynamics of the environment while our method learns the latent transitions and achieves good performance across all domains.	237
F.1	Pie chart of RBM-1M dataset types. Full table with individual dataset details in Table F.4.	245
F.2	Robometer model architecture. A VLM backbone processes a language instruction with one or two trajectories. For both single and paired inputs, per-frame hidden states of the first trajectory are passed to MLP heads predicting dense task progress and binary success. For paired trajectories, a preference token is used with a preference head to select the trajectory that better satisfies the instruction. The model is trained with joint progress, success, and preference objectives.	263
F.3	Qualitative analysis of Robometer on a long-horizon task. With sufficiently detailed task instructions, Robometer correctly assigns increasing rewards across multiple stages of the task.	269
F.4	Failure Detection OOD confusion matrices with ternary ground truth and binary prediction. Rows indicate ground-truth execution outcomes (<i>failure</i> , <i>sub-optimal</i> , <i>success</i>), while columns indicate binary predictions (<i>predicted failure</i> vs. <i>predicted success</i>). <i>Suboptimal</i> trajectories correspond to executions that make partial progress but do not complete the task. Suboptimal trajectories are treated as failures in our quantitative evaluation, as emphasized with the horizontal orange divider in the above confusion matrices. Color intensity reflects the number of trajectories.	277

F.5	Irreversible failures. Terminal events such as drops or spills cause a sharp regression in predicted task progress, which our model reliably flags as failures shortly after the event.	279
F.6	Semantic failures. The robot executes smooth and physically plausible trajectories but violates the task instruction, resulting in persistently low predicted progress and failure detection without an abrupt terminal event.	280
F.7	Insufficient-progress failures. Although the robot continues to move, predicted progress stagnates or oscillates without converging to success, enabling detection of failures caused by stalling or premature termination. Execution stalls with plateaued progress in this example.	280
F.8	Data filtering & retrieval scene configuration from all cameras.	281
F.9	Model-Based RL with Robometer integrated into DreamZero [320]. In this cluttered scene, Robometer improves DreamZero’s performance from 20% success rate to 70%.	288

Abstract

Children learn to perform skillful tasks through a combination of watching others, imitating them, being gently corrected, and refining their behavior through practice and environmental feedback. Much of this process depends on natural interaction with other people: a glance that indicates what to attend to, a hand motion that conveys intent, a short verbal correction, or a preference expressed between two options. Over time, this guidance is internalized, and learning increasingly comes from the environment itself as the learner acts, observes consequences, and adjusts. A core premise of this dissertation is that robot learning should follow a similar trajectory, drawing on cheap and natural forms of human guidance early on while relying on autonomous experience to refine behavior after deployment.

Modern robot learning has made significant strides by scaling imitation learning on large, curated demonstration datasets collected through teleoperation. These efforts have produced increasingly capable pretrained policies, analogous to foundation models in language and vision. However, teleoperated demonstrations are expensive, slow to collect, and difficult to scale to the long tail of tasks, environments, and user preferences encountered in real deployments. At the same time, a fixed policy trained on such data remains brittle: when deployed in homes and workplaces, robots inevitably encounter distribution shifts that no amount of pretraining can fully anticipate. Closing this gap requires rethinking where supervision comes from and how robots continue to learn once they leave the training distribution.

The first part of this dissertation develops methods for *scalable supervision from natural human signals*. Rather than depending on teleoperated demonstrations, I study how robots can

learn from lighter-weight cues that people produce during ordinary interaction. ViSaRL uses human attention, expressed as gaze or sparse saliency, as an inductive bias for visual reinforcement learning. HAND enables users to specify tasks through short RGB videos of their own hand motions, which are then used to retrieve matching behaviors from a large offline robot dataset. CLAM leverages action-free Internet video by learning latent actions in a self-supervised manner, grounding them with a small amount of labeled data so that policies can benefit from large-scale human video. PLANTAIN introduces a plan-based inference framework that surfaces intermediate plans before execution, allowing users to inspect and revise the model’s intent and making language-conditioned behavior more controllable.

The second part of this dissertation studies *autonomous improvement under distribution shift*, where the learner itself must close the loop with its environment. DynaMITE-RL formulates adaptation as a meta-reinforcement learning problem over task dynamics, allowing a robot to infer and respond to changing conditions from its own experience across episodes. Robometer builds a video-language reward model that provides dense task progress and success signals, trained from both per-frame supervision and trajectory-level comparisons. Together, these methods give robots a way to learn from suboptimal trajectories, near-misses, and partial progress, rather than treating deployment as a fixed inference-time procedure.

Across both parts, these contributions argue for two complementary principles: supervision at training time should be cheap, natural, and aligned with how people already communicate, and adaptation at deployment time should be structured, enabling robots to continue learning from their own interaction with the world. The path toward reliable robots in open-world settings lies not in more expensive demonstrations alone, but in combining lightweight human guidance with autonomous improvement so that systems remain both steerable by non-experts and capable of becoming more competent over time.

INTRODUCTION

Chapter 1

Introduction

This dissertation studies a central question:

How can we design robot learning systems that are both easy for people to guide and able to continually improve after deployment?

Modern robot learning is increasingly organized around the same paradigm that powers progress in language and vision: pretrain on large and diverse datasets, then adapt the resulting model to specific settings [33, 35, 40, 155]. One can imagine robots trained in data-rich settings like factories, research labs, or large-scale collection pipelines, where experience can be gathered systematically through teleoperation or scripted policies [41, 61, 230]. Such pretraining can equip robots with a broad set of perceptual and motor skills, enabling them to perform a range of tasks out of the box.

However, pretraining alone does not solve the problem of real-world deployment.

The challenge emerges when robots enter human environments such as homes and workplaces. These settings are inherently open-ended: tasks vary across users, preferences are often underspecified, and small variations in the environment can lead to unexpected behavior [10, 240]. As a result, a robot may capture the general structure of a task but still struggle to perform it reliably in new settings.

A common approach to adapting robots in these settings is to collect expert demonstrations. When the robot fails, additional demonstrations are collected to correct its behavior. While effective, this paradigm does not scale cleanly. Teleoperation is expensive, slow, and requires specialized hardware and training, making it difficult for end users to provide [94, 339]. Each new task variation or environment can require additional supervision, and even with more demonstrations, policies trained purely by imitation remain brittle, since distribution shift between training and execution causes small errors to compound into large failures [7, 279].

For robots to be useful in everyday settings, they must go beyond imitation. Pretraining and demonstrations provide a strong initialization, but they are not sufficient to achieve the level of reliability required for real-world use. Systems must be able to learn from interaction, recover from failures, and improve over time. Reinforcement learning and continual adaptation provide a way to use interaction as a training signal, allowing systems to improve from failures and become more reliable over time [106, 257].

At the same time, these systems must remain steerable by non-expert users. In real deployments, people cannot be expected to teleoperate robots or provide large quantities of high-quality demonstrations. Instead, robots should be teachable through lighter-weight and more natural forms of guidance: attention, motion cues, language, corrections, and preferences [16, 30].

This dissertation explores both sides of this problem. On one hand, it investigates how to leverage scalable forms of human supervision that are cheaper and more natural than expert demonstrations. On the other hand, it studies how robots can continue improving under distribution shift, using richer feedback and interaction to increase reliability over time.

These ideas are developed through two complementary directions:

1. **Scalable supervision from human signals**, which reduces dependence on expert demonstrations by leveraging natural human feedback and large-scale video.
2. **Autonomous improvement under distribution shift**, which enables robots to adapt after deployment through reward modeling, reinforcement learning, and temporal structure.

Chapter 2 introduces the formal foundations used throughout the thesis, including Markov

decision processes, imitation learning, behavior cloning, and reinforcement learning.

1.1 Scalable Supervision from Human Signals

The first part of this dissertation focuses on reducing the burden of supervision required to teach robots new behaviors. If pre-trained robots are to be useful in real-world settings, they must be steerable through interfaces that are natural for non-expert users.

Human attention as guidance. In Chapter 3 (ViSaRL), I study how attention signals such as gaze or sparse click-based saliency can guide learning. These signals are significantly easier to collect than full demonstrations, yet still convey what matters in a scene. ViSaRL shows that attention can serve as an inductive bias for representation learning, improving robustness and data efficiency in visual reinforcement learning. This work was published at IROS 2024 [188].

Human motion as a natural interface. In Chapter 4 (HAND), I study how robots can learn from human hand demonstrations without requiring teleoperation or action labels. Instead, we extract 2D hand path traces from RGB video and use them as a query to retrieve corresponding robot behaviors from a large offline dataset. This allows users to specify tasks through simple hand motions, which are significantly easier to provide than teleoperated demonstrations. This work will appear at ICRA 2026 [120].

Internet video as scalable supervision. Robot interaction data is fundamentally limited by the cost of collection. In contrast, Internet video provides a vast source of demonstrations of everyday activities. In Chapter 5 (CLAM), I leverage this observation by learning from observation-only video, where latent actions are learned in a self-supervised manner and grounded with a small amount of labeled data. This enables robots to benefit from large-scale human video without requiring direct action annotation. This work is available as a preprint [189].

Planning for controllable interaction. Natural language task specifications are often ambiguous, especially for large language models that must reason over multiple possible interpretations. When applied to robotics, this ambiguity can translate into incorrect or unintended actions. In Chapter 6 (PLANTAIN), I introduce a plan-based inference framework that surfaces interme-

diate plans before execution. This allows users to inspect or revise the model’s intent prior to commitment, enabling more controllable behavior. This work was accepted as a Spotlight at ICML 2026 [190].

Together, these works emphasize that scalable robot learning should leverage the broader space of human signals rather than relying exclusively on expert demonstrations.

1.2 Autonomous Improvement under Distribution Shift

Even with scalable supervision, deployment introduces new challenges. Robots must operate under distribution shift: new environments, new tasks, and new failure modes that were not encountered during training. In these settings, a fixed policy is insufficient.

The second part of this dissertation studies how robots can improve through experience after deployment.

Learning to adapt over time. In Chapter 7 (DynaMITE-RL), I study meta-reinforcement learning as a framework for adaptation. Rather than treating adaptation as a separate procedure, this work models it as part of the learning problem itself. By capturing temporal structure across episodes, the system can respond to changing environments and update its behavior more effectively. This work was published at NeurIPS 2024 [187].

Reward models for dense feedback. In Chapter 8 (Robometer), I study how to construct learning signals that remain useful across tasks and environments. Robometer is a video-language reward model that predicts dense task progress and success, trained using both per-frame supervision and trajectory-level comparisons. This provides a scalable source of feedback for reinforcement learning and post-training in real-world settings. This work was accepted at RSS 2026 [334].

BACKGROUND

Chapter 2

Background: Reinforcement Learning, Imitation Learning, and Offline RL

This chapter collects standard definitions and notation for *reinforcement learning* (RL), *imitation learning* (IL), and *offline reinforcement learning*. We follow the Markov decision process (MDP) formulation used throughout the RL literature [179, 272].

2.1 Markov decision processes and reinforcement learning

We consider an agent interacting with an environment modeled as a *Markov decision process* (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0),$$

where $\mathcal{S}$ and $\mathcal{A}$ denote state and action spaces, $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition kernel mapping each state–action pair (s, a) to a distribution over next states $s' \sim P(\cdot \mid s, a)$, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor, and $\rho_0 \in \Delta(\mathcal{S})$ is an initial state distribution with $s_0 \sim \rho_0$ [272].

A *policy* $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ specifies a distribution over actions $a_t \sim \pi(\cdot \mid s_t)$ at each time t . Starting from s_0 , rolling out π induces a distribution over trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ with $r_t = r(s_t, a_t)$ and $s_{t+1} \sim P(\cdot \mid s_t, a_t)$. The (infinite-horizon, discounted) return along a

trajectory is

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

and the usual RL objective is to maximize expected return

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right],$$

where the expectation is over randomness in the initial state, transitions, and policy.

The *state–action value function* (Q-function) of π is

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a],$$

and the optimal Q-function is $Q^*(s, a) = \sup_\pi Q^\pi(s, a)$. The optimal Q-function satisfies the Bellman optimality equation

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} \left[\max_{a' \in \mathcal{A}} Q^*(s', a') \right].$$

Online RL algorithms improve π using data collected by the learner’s own interaction with the environment (on-policy or off-policy), typically via value estimation, policy gradients, or actor–critic updates [272].

2.2 Imitation learning

Imitation learning (also called *learning from demonstration*) uses example behavior from a reference source (often a human or expert policy) to train a policy without an explicit reward signal at training time [7]. A common setting is a dataset of state–action pairs

$$\mathcal{D}_{\text{demo}} = \{(s_i, a_i)\}_{i=1}^N,$$

collected (approximately) from an expert policy π_E .

Behavior cloning (BC) treats imitation as supervised learning: one chooses a parametric policy π_θ and minimizes a loss that matches expert actions, for example the negative log-likelihood

$$\mathcal{L}_{\text{BC}}(\theta) = -\mathbb{E}_{(s, a) \sim \mathcal{D}_{\text{demo}}} [\log \pi_\theta(a \mid s)]$$

for discrete actions, or a squared-error loss on continuous actions. Because errors compound under the learned policy’s state distribution, BC can suffer from *covariate shift* relative to the

expert’s state visitation; a variety of interactive and iterative methods address this issue [7]. Other imitation approaches estimate rewards or objectives from data (e.g., inverse optimal control and inverse RL) [227].

2.3 Offline reinforcement learning

Offline RL (also called *batch RL*) assumes access to a fixed dataset of transitions

$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N,$$

collected by one or more *behavior* policies, without further online interaction during training [91, 179]. The goal remains to learn a policy π that maximizes expected discounted return in the MDP, but optimization must rely entirely on $\mathcal{D}$.

A central difficulty is *distribution shift*: policies queried during learning may visit (s, a) pairs that are poorly covered in $\mathcal{D}$, so value estimates based on bootstrapping through P can become optimistic and unstable. Modern offline RL methods therefore regularize learning toward the behavior policy, conservatively penalize out-of-distribution actions in value learning, or otherwise constrain policy improvement [160, 179]. In this dissertation, offline RL is used in conjunction with learned reward models and meta-training, building on this setting while addressing nonstationarity and feedback during deployment.

I

Scalable Supervision from Human Signals

Part I: Scalable Supervision from Human Signals

The first part of this dissertation focuses on the question of *where supervision comes from*. Behavior cloning on teleoperated demonstrations has emerged as the dominant paradigm for training robot policies [33, 94, 230, 338], but it places a heavy burden on the demonstrator. Teleoperation requires specialized hardware, is slow to collect, and assumes that a skilled operator is always available to drive the robot through the exact behavior the user wants. This is practical in well-resourced research labs, but it is not how non-experts will teach robots in homes, warehouses, or clinics.

The central premise of this part is that supervision should be *cheap, natural, and aligned with how people already communicate intent*. Before people pick up a teleoperation rig, they can glance at an object, trace a motion with their hand, show a short video of the task, or describe it in language. Each of these signals encodes useful information about what the robot should attend to, how it should move, and what the user wants. Used well, they can reduce the number of full demonstrations required, expand the pool of data the robot can learn from, and make interfaces accessible to users who cannot or will not teleoperate.

The four chapters in this part each explore one such signal. Chapter 3 treats human attention, expressed as gaze or sparse click-based saliency, as an inductive bias for visual reinforcement learning. Chapter 4 uses short RGB videos of a user’s hand motion as a retrieval query over a large offline robot dataset, letting users specify tasks without teleoperating a single trajectory.

Chapter 5 turns to action-free Internet video, learning latent actions in a self-supervised manner so that policies can benefit from observation-only data at scale. Chapter 6 addresses the ambiguity of natural language by surfacing intermediate plans at inference time, giving users a handle to inspect and redirect the model’s intent before execution.

Across these chapters, the shared argument is that the right question is not only “how do we collect more demonstrations,” but “what is the minimum signal a person needs to provide for the robot to behave the way they want?”

CHAPTER 3

Chapter 3

ViSaRL: Visual Reinforcement Learning Guided by Human Saliency

3.1 Introduction

Studies in neuroscience [68] show that humans utilize selective attention to focus on task-relevant information for efficiently processing and understanding complex visual scenes [135]. We employ selective attention when performing everyday pick-and-place tasks to identify the target objects, focus on the grasp points, and execute precise hand-eye coordination. We hypothesize that saliency maps capturing human visual attention is a useful signal to process visual observations for AI agents. In this chapter, we investigate whether *human* visual attention helps *agents* perform tasks.

A key ingredient in solving visual control tasks is to learn visual representations that capture useful features of the sensory input to simplify the decision-making process. Many works in the deep reinforcement learning (RL) community have proposed to learn such representations through various self-supervised objectives including contrastive learning [170] and data augmentation [171]. By contrast, we focus on self-supervision using *saliency* as additional human domain knowledge to inform the representation of task-relevant features in the visual input while filtering out perceptual noise.

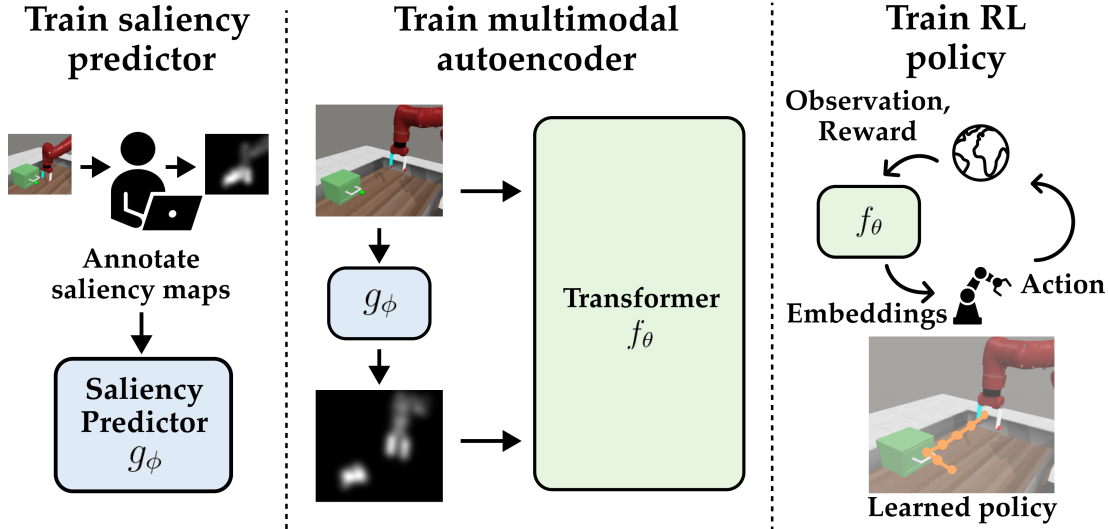


Figure 3.1: ViSaRL trains a saliency prediction model from a few human-annotated saliency maps. This model is used to augment an offline image dataset with saliency. A visual encoder is pretrained with the dataset and used during downstream policy learning to generate latent representations of the agent’s observations.

We present **Visual Saliency Reinforcement Learning (ViSaRL)**, a general approach for incorporating human-annotated saliency maps as an inductive bias for learned visual representations. The key idea of ViSaRL is to train a visual encoder using both RGB and saliency inputs and an RL policy that operates over lower dimensional image representations as shown in Figure 3.1. By using a multimodal autoencoder trained using a self-supervised objective, our learned representations attend to the most salient parts of an image for downstream task learning making them robust to visual distractors. To circumvent the expensive process of manually annotating saliency maps, we train a state-of-the-art saliency predictor using only a few human-annotated examples to pseudo-label RGB observations with saliency.

We evaluate ViSaRL on a diverse set of challenging continuous control tasks in the DeepMind Control (DMC) suite [286] and robot manipulation tasks in Meta-World [325] and a real robot. Our method improves in sample-efficiency and robustness over state-of-the-art vision-based RL methods across all environments. Remarkably, ViSaRL nearly doubles the task success rate on a real-robot.

Our contributions can be summarized as follows:

1. We propose ViSaRL, a framework for incorporating human-annotated saliency maps to learn robust representations for visual control tasks;
2. We present approaches for utilizing saliency information in both CNN and Transformer encoders; and
3. We conduct extensive experiments that demonstrate ViSaRL consistently outperform prior state-of-the-art methods for visual control tasks both in simulation and on a real robot.

3.2 Related Work

Different forms of human data can be leveraged when solving control tasks. Researchers have created various interfaces to collect different data modalities from humans such as reward sketches [45], feature traces [34], scaled comparisons [300], and abstract trajectories [275]. Attention saliency maps, in contrast, do not require humans to work with abstract concepts like rewards and task features, and do not require watching and comparing lengthy trajectories.

Saliency Maps. Saliency maps approximate which parts of an image tend to attract human visual attention, corresponding to where the human eye would likely fixate when viewing an image [285]. Saliency maps have been used in both computer vision and machine learning for various applications including activity recognition [297], question answering [69], and object segmentation [182]. The explainable AI community uses saliency maps to understand how a model is making its predictions and to identify the most informative regions of an image for a particular task [220, 265, 337]. Most existing works explore using saliency maps only as tools for interpretation [9, 250]. For example, Atrey et al. [9] and Rosynski et al. [250] use saliency maps to rationalize and explain the actions of RL agents in Atari games. Boyd et al. [37] show saliency maps encoding prior human knowledge enable better generalization of deep learning models.

Bertoin et al. [27] uses neural network saliency in a self-supervised regularization objective to encourage better visual representations. We do not use a model’s saliency, but rather human saliency to identify salient regions of the input image and distill this knowledge into the visual representation.

User Interfaces for Human Saliency. ViSaRL needs a small number of human-annotated saliency maps to bootstrap the saliency prediction network. Prior work used superpixel segmentation [2] to first divide each image into segments, and then asked humans to click on the segments that are salient [337]. However, that method requires manually checking and combining the segments that belong to the same object before showing the images to annotators, burdening system designers. As an alternative, Boyd et al. [36] used interfaces where the annotators created binary masks by simply clicking on images. We employ a similar but simpler interface: an annotator clicks on the salient parts of the image, and a Gaussian kernel is applied around selected pixels to achieve smooth saliency maps shown in Figure 3.4.

Representation Learning for RL. Saliency maps are representations of the environment that carry domain knowledge about which regions of the visual input are important for the downstream task. Such representations are crucial in RL because they enable agents to tractably deal with high-dimensional image observation spaces.

Prior works have shown self-supervised learning with data augmentation helps achieve good performance in image-based RL. Contrastive Unsupervised RL (CURL) [170] employs a contrastive learning objective as an auxiliary loss to learn representations for off-policy RL. RL with Augmented Data (RAD) [171] use simple image augmentations such as random cropping and color jittering as regularization to learn representations invariant to visual perturbations. ViSaRL does not use data augmentation directly in the value function or policy update. Instead, saliency augmentation is introduced during the visual encoder pretraining phase.

Nair et al. [224] and Karamcheti et al. [146] propose to combine internet scale language and vision datasets to learn visual representations applicable across all robot tasks. While they focus on learning general visual representations, ViSaRL augments small task-specific datasets with saliency information to improve pretrained visual representations.

Sax et al. [254] demonstrated that mid-level visual representations such as surface normals or depth predictions from RGB images removes unimportant information and captures useful invariances about the visual world leading to better success on downstream RL tasks. Similar to

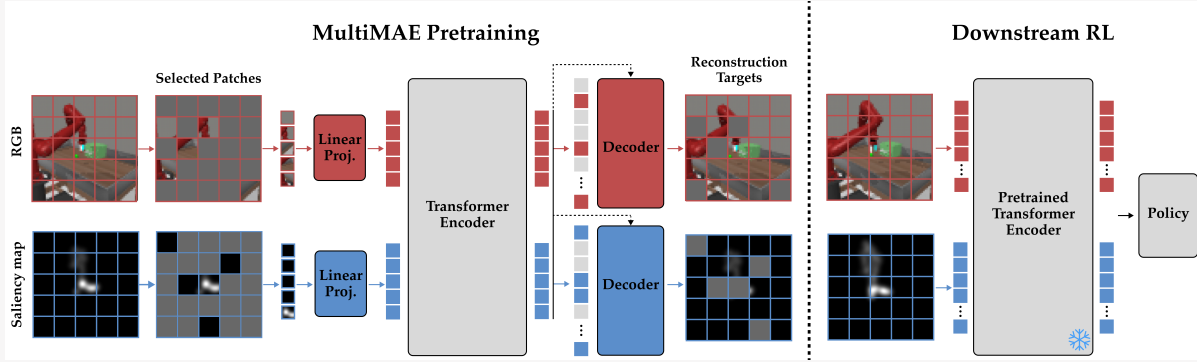


Figure 3.2: ViSaRL. We pretrain a MultiMAE [12] Transformer on a dataset of paired images and saliency maps. MultiMAE employs a self-supervised objective in which masked patches for both input modalities are reconstructed given only the visible patches. The pretrained model is frozen and used for extracting representations during task learning. There is no input masking during downstream RL.

Sax et al. [254], ViSaRL utilizes saliency maps as a mid-level feature. However, we empirically show that our approach for incorporating the saliency information into the visual representation improves task performance over other mid-level features including depth and surface normals.

3.3 Visual Saliency-Guided Reinforcement Learning

We propose ViSaRL, a simple approach for incorporating human-annotated saliency to learn representations for visual control tasks. ViSaRL can be implemented on top of any standard RL algorithm for learning a policy. It aims to learn representations that encode useful task-specific inductive biases from human saliency maps. ViSaRL consists of three learned components: a saliency predictor g_ϕ , an image encoder f_θ , and a policy network π_ψ shown in Figure 3.1. We will elaborate on each component in the following sections.

Saliency Predictor. Saliency maps highlight regions in an image likely to capture human attention or are considered crucial for a given task. Having a human expert annotate saliency maps for every image observation is impractical and not scalable to complex domains. To alleviate the burden of manual annotations, we propose to learn a saliency network using only a few hand-annotated examples of saliency maps collected using a custom user interface.

Formally, given an input RGB image observation, $I \in \mathbb{R}^{H \times W \times C}$, a saliency predictor g_ϕ maps an

input image I to a continuous saliency map $M = g_\phi(I) \in [0, 1]^{H \times W}$ highlighting important parts of the image for the downstream task. We use a state-of-the-art saliency model, Pixel-wise Contextual Attention network (PiCANet) [199]. PiCANet uses global and local pixel-wise attention modules to selectively attend to informative context. Global attention can attend to backgrounds for foreground objects while local attention can attend to regions that have similar appearance. The mixture of attention at different scales allows for more homogeneous and consistent saliency predictions. We emphasize that our method is agnostic to the choice of saliency model.

Pretraining Visual Representation. We use our trained g_ϕ to pseudo-label an offline image dataset collected using any behavior policy (random, replay buffer, expert demonstrations, etc.) with saliency maps. We then use the paired image and saliency dataset to pretrain an image encoder, f_θ . We experiment with two models for our backbone visual encoder, CNN and Transformer, and investigate different techniques for augmenting each with saliency input. To add saliency to a CNN, we can use saliency as a continuous mask or simply add it as an additional channel per pixel. For a Transformer encoder, we pretrain the model with saliency as an additional input using a masked reconstruction objective.

Masked autoencoders (MAE) [114] are an effective and scalable approach for learning visual representations. MAE masks out random patches of an image and reconstructs the masked patches using a Vision Transformer (ViT) [74]. An image $I \in \mathbb{R}^{H \times W \times C}$ is processed into a sequence of 2D patches $h \in \mathbb{R}^{K \times (P^2 C)}$ where P is the patch size and $K = HW/P^2$ is the number of patches. A subset of these patches are randomly masked out with a masking ratio of m . Only the *visible, unmasked patches* are used as input to the ViT encoder. Masking reduces the input sequence length and encourages learning global, contextualized representations.

The image patches are embedded via a linear projection and added to positional embeddings. The resulting tokens are processed via a series of Transformers. Finally, a ViT decoder reconstructs the original input by processing all of the tokens including the encoded visible patches and placeholder mask tokens. Following He et al. [114], we set a high masking ratio $m = 0.75$ and a heavy-encoder, light-decoder architecture.

Algorithm 1 Visual Saliency-Guided RL

- 1: **Input:** $\text{env}, \phi, \psi, \theta$ randomly initialized parameters
 - 2: Collect image dataset $\mathcal{D}$ with any behavioral policy π_B
 - 3: Annotate N random frames from $\mathcal{D}$ with saliency
 - 4: Train g_ϕ on $\{(I, M)\}_{i=1}^N$ using PiCANet loss
 - 5: Annotate the full dataset $\mathcal{D} = \{(I, g_\phi(I))\}_{i=1}^N$
 - 6: Train f_θ using masked reconstruction
 - 7: **for** every environment step **do** $\triangleright$ RL Training
 - 8: Select action $a \sim \pi_\psi(f_\theta(o, g_\phi(o)))$
 - 9: Optimize $\mathcal{L}_{RL}$ with respect to ψ
-

MultiMAE for Encoding Saliency. The standard MAE architecture is limited to processing just RGB modality. We propose to incorporate saliency using the MultiMAE [12] architecture shown in Figure 3.2. MultiMAE extends MAE to encode multiple input modalities in a way that these modalities are contributing synergistically to the resulting representation. Specifically, MultiMAE uses a different linear projection and decoder for each input modality. A cross attention layer is used in each decoder to incorporate information from the encoded tokens of other modalities. Crucially, MultiMAE’s pretraining objective requires the model to perform well in both the original MAE objective of RGB in-painting and cross-modal reconstruction, resulting in a stronger cross-modal visual representation.

Downstream Policy Learning. After pretraining the MultiMAE model, we freeze the encoder and use it to compute latent representations of environment observations for policy training. ViSaRL is not only compatible with online RL algorithms such as Soft-Actor Critic (SAC) [106] in which the agent learns through environment interactions but also imitation learning from expert demonstrations. Image inputs are not masked during policy learning. We average the patch embeddings to generate a global image representation. The full procedure for ViSaRL is summarized in Algorithm 1.

3.4 Experiment Setup

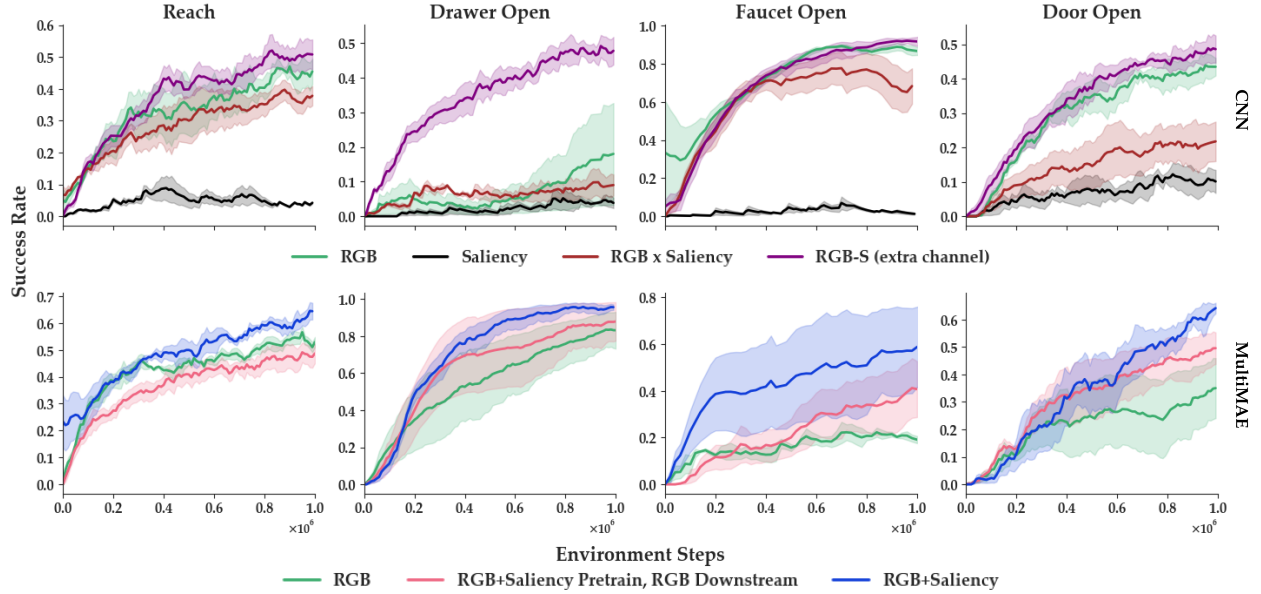


Figure 3.3: Learning curves for four robot manipulation tasks in Meta-World evaluated by task success rate. **(Top)** CNN encoder methods. **(Bottom)** Transformer encoder methods. We select tasks that require manipulating small objects with different motions such as a pushing, pulling, and reaching. The solid lines represent the mean and shaded region the standard error across three seeds.

To demonstrate the effectiveness of using human-annotated saliency information to enhance visual representations for task learning, we show quantitative results of our approach with two different encoder backbones, CNN and Transformer, across multiple simulated environments including the Meta-World manipulation [325] and DMC benchmarks [27] and real-robot manipulation with a Kinova Jaco 2 arm. We train the downstream policy using SAC [106] for the simulation experiments and behavioral cloning with expert demonstrations for the real robot experiments.¹

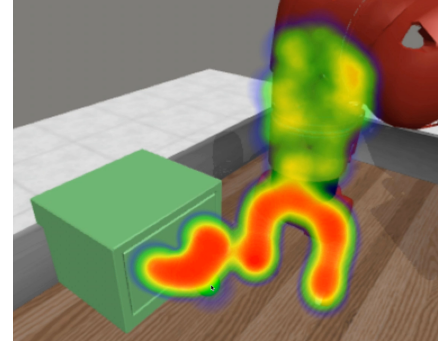


Figure 3.4: Annotation interface. Click-based saliency collection (Gaussians at clicked pixels). Warmer colors denote more salient regions such as the drawer handle and the robot end-effector.

Annotation interface. We created a simple user interface to collect saliency map annotations, shown in Figure 3.4: an annotator clicks pixels in the image that they think are relevant for performing the given downstream task, and the interface places a Gaussian centered at each click with $\sigma = 10$ on an input image of resolution $256 \times 256 \times 3$. For each task, we sample frames

¹The code implementation for reproducing the results and additional analysis can be found on: <https://liralab.usc.edu/visar/>.

Table 3.1: Success rate on four Meta-World manipulation tasks averaged across 50 rollouts and 3 seeds for the CNN and MultiMAE visual encoder backbones. Text in **maroon** indicates the best performing method per task.

Method	Reach	Drawer Open	Faucet Open	Door Open	Avg.
<i>CNN</i>					
RGB	0.40 $\pm$ 0.12	0.18 $\pm$ 0.25	0.82 $\pm$ 0.02	0.42 $\pm$ 0.04	0.46 $\pm$ 0.11
Saliency	0.04 $\pm$ 0.02	0.04 $\pm$ 0.02	0.02 $\pm$ 0.02	0.10 $\pm$ 0.06	0.05 $\pm$ 0.03
RGB $\times$ Saliency	0.38 $\pm$ 0.05	0.10 $\pm$ 0.04	0.72 $\pm$ 0.16	0.22 $\pm$ 0.10	0.36 $\pm$ 0.08
RGB(S)	0.52 $\pm$ 0.08	0.48 $\pm$ 0.06	0.86 $\pm$ 0.02	0.48 $\pm$ 0.06	0.58 $\pm$ 0.05
<i>MultiMAE</i>					
RGB	0.50 $\pm$ 0.02	0.84 $\pm$ 0.02	0.18 $\pm$ 0.05	0.36 $\pm$ 0.18	0.48 $\pm$ 0.12
RGB+Saliency (PO)	0.48 $\pm$ 0.06	0.88 $\pm$ 0.04	0.40 $\pm$ 0.20	0.52 $\pm$ 0.08	0.57 $\pm$ 0.10
RGB+Saliency (Ours)	0.62 $\pm$ 0.06	0.94 $\pm$ 0.04	0.62 $\pm$ 0.16	0.64 $\pm$ 0.02	0.65 $\pm$ 0.07

from environment rollouts and present them to the annotator in sequence, collecting $N = 30$ annotated images per task in simulation and $N = 10$ for real-robot experiments.

3.5 Simulation Experiments

Figure 3.3 and Table 3.1 summarize our main findings on 4 Meta-World robot manipulation tasks and 5 DMC tasks using CNN and Transformer backbones. We compare against two state-of-the-art methods for visual representation learning: CURL [170], a contrastive representation learning method and RAD [171], a method to combine various image augmentations to induce visual invariances in the learned representations.

Saliency input improves downstream task success rates. Incorporating saliency improves the task success rate in Meta-World using CNN and Transformer encoders by 13% and 18% respectively over the next best baseline. For DMC environments, we observe a 256% relative improvement in average return when using saliency input. Our Transformer encoder results in an average 4% relative improvement in environment returns across all tasks over the next best baseline with a 7.5% improvement in Cartpole Swing.

Table 3.2: Average return on DMC-GB tasks for the color and video evaluation settings.

	Task	CURL	RAD	RGB + Saliency (Ours)
color	Walker Walk	645 $\pm$ 55	636 $\pm$ 33	823 $\pm$55
	Cartpole Swing	668 $\pm$ 74	763 $\pm$ 29	870 $\pm$21
	Ball Catch	565 $\pm$ 160	727 $\pm$ 87	962 $\pm$14
	Finger Spin	781 $\pm$ 139	789 $\pm$ 160	823 $\pm$102
video	Walker Walk	572 $\pm$ 121	595 $\pm$ 85	756 $\pm$42
	Cartpole Swing	418 $\pm$ 72	434 $\pm$ 58	730 $\pm$32
	Ball Catch	402 $\pm$ 169	520 $\pm$ 44	802 $\pm$78
	Finger Spin	612 $\pm$ 55	588 $\pm$ 82	702 $\pm$83

3.5.1 CNN Encoder

We follow the CNN implementation used in prior work [27, 171] and compare several methods for incorporating saliency. In each approach, the CNN encoder and policy are trained jointly but take different inputs.

A saliency channel achieves the best task success rate for CNN encoder. In Table 3.1, we find that naive ways of utilizing saliency, such as using saliency directly as input the policy (**Saliency**), are unable to achieve good performance on the task. We hypothesize that the saliency map alone is not sufficient to infer the exact orientation of the end-effector position critical for fine control. Supporting this hypothesis, we find that using saliency to mask the RGB observation (**RGB $\times$ Saliency**) achieves higher task success rate than **Saliency**, but is still worse than providing the raw RGB input (**RGB**). Although masking should help the encoder identify the important image features, it may still be nontrivial for the encoder to differentiate between similarly masked observations. Lastly, we find that incorporating saliency as an additional channel to the RGB input (**RGB(S)**) improves task success rate by more than 10% across all tasks. We hypothesize that the CNN encoder is able to utilize the saliency information to more effectively associate the observed rewards to the relevant features in the image.

3.5.2 MultiMAE Transformer

We compare MultiMAE representations pretrained with RGB only (**RGB**) and both RGB and saliency (**RGB+Saliency (PO)**, **RGB+Saliency (Ours)**). **RGB+Saliency (PO)** uses saliency only during pretraining while **RGB+Saliency (Ours)** uses saliency in both pretraining and downstream RL.

Training encoder with saliency improves RGB-only success rates at inference time. Even without saliency input during downstream RL, using saliency as an additional input modality during pretraining still improves downstream performance on 3 of the 4 tasks. Except for the Reach task, where performances are similar, **RGB+Saliency (PO)** achieves better success rate than **RGB**, with an average absolute gain of 10% across tasks.

Using saliency in both pretraining and inference yields the best performance. We compare the full ViSaRL method (**RGB+Saliency (Ours)**) to pretraining using only the RGB images (**RGB**) in Table 3.1 demonstrating that multimodal pretraining with saliency information significantly outperforms single modality pretraining by at least a 10% margin across all tasks. Notably, **RGB** achieves only 19% success on Faucet Open, while our approach solves the task with 62% success rate. Using saliency as an input for both pretraining and downstream RL (**RGB+Saliency (Ours)**) improves task success rate over **RGB+Saliency (PO)** because there are new observations during online training that were not in the pretraining dataset.

ViSaRL representations generalize to unseen environments. We evaluate the generalizability of our learned representations on the challenging *random colors* and *video backgrounds* benchmark from DMControl-GB [111]. In DMControl-GB, agents trained in the original environment are evaluated on their generalization to the same environment with visually perturbed backgrounds using randomized color and video overlays (see Figure A.1). ViSaRL significantly outperforms the baselines across all tasks as shown in Table 3.2, with an average 19% and 35% relative improvement respectively for the `color` and `video` settings.

Human-annotated saliency improves performance compared to depth and surface normals. We conduct ablation experiments to compare saliency versus other mid-level input modal-

Table 3.4: Human-annotated saliency versus depth and predicted surface normals as additional MultiMAE inputs. *SN* denotes *surface normal*.

Task	Saliency	RGB	RGB + Depth	RGB + SN
Reach	✗	0.50 $\pm$ 0.02	0.43 $\pm$ 0.07	0.46 $\pm$ 0.04
	✓	0.62 $\pm$ 0.06	0.58 $\pm$ 0.04	0.64 $\pm$ 0.06
Drawer Open	✗	0.82 $\pm$ 0.02	0.76 $\pm$ 0.06	0.80 $\pm$ 0.04
	✓	0.94 $\pm$ 0.04	0.90 $\pm$ 0.04	0.92 $\pm$ 0.04
Faucet Open	✗	0.18 $\pm$ 0.04	0.22 $\pm$ 0.04	0.24 $\pm$ 0.04
	✓	0.62 $\pm$ 0.16	0.54 $\pm$ 0.06	0.58 $\pm$ 0.10
Door Open	✗	0.36 $\pm$ 0.18	0.28 $\pm$ 0.14	0.34 $\pm$ 0.10
	✓	0.64 $\pm$ 0.02	0.62 $\pm$ 0.04	0.58 $\pm$ 0.04

ities such as depth and surface normals proposed by Sax et al. [254]. We substitute saliency with these other modalities as input to the MultiMAE.

In Table 3.4, we observe that neither depth nor surface normal features alone improves task success over just using RGB image input. By contrast, adding saliency as an additional modality consistently improves task success suggesting that human-annotated saliency information can help learn better visual representations com-

Table 3.3: Adding saliency improves performance across tasks and modalities for MultiMAE. *SN* denotes *surface normal* (columns match Table 3.4).

Task	RGB + Depth	RGB + SN
Reach	0.43 $\rightarrow$ 0.58	0.46 $\rightarrow$ 0.64
Drawer Open	0.76 $\rightarrow$ 0.90	0.80 $\rightarrow$ 0.92
Faucet Open	0.22 $\rightarrow$ 0.54	0.24 $\rightarrow$ 0.58
Door Open	0.28 $\rightarrow$ 0.62	0.34 $\rightarrow$ 0.58

pared to other input modalities. Table 3.3 reports the corresponding improvements when saliency is enabled for the RGB+depth and RGB+surface-normal configurations.

3.6 Real Robot Experiments

We use a Kinova Jaco 2 (6-DoF) robot arm with a 1-DoF gripper. The observation space consists of a front-view image ($224 \times 224 \times 3$) from a Logitech webcam and proprioceptive information.

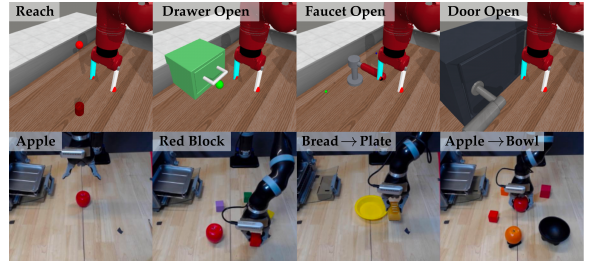
Table 3.5: RGB vs. RGB+Saliency (ours) with absolute improvements in task success.

Task	RGB $\rightarrow$ RGB+Saliency	Absolute Δ
Reach	0.49 $\rightarrow$ 0.62	0.13
Drawer Open	0.83 $\rightarrow$ 0.94	0.11
Faucet Open	0.19 $\rightarrow$ 0.61	0.42
Door Open	0.35 $\rightarrow$ 0.64	0.29
Average	0.47 $\rightarrow$ 0.65	0.18

Table 3.6: Human-eval successes (trials out of 10) with saliency improvements on real-robot tasks.

Task	RGB $\rightarrow$ RGB+Saliency
Apple	6/10 $\rightarrow$ 8/10
Red Block	4/10 $\rightarrow$ 7/10
Bread $\rightarrow$ Plate	3/10 $\rightarrow$ 6/10
Apple $\rightarrow$ Bowl	1/10 $\rightarrow$ 6/10

We consider four tabletop manipulation tasks shown in Figure 3.5. In two of these tasks, we purposefully include distractor objects to evaluate the robustness of our learned representations to scene variations. We collected 10 demonstrations per task, resulting in an offline imitation learning dataset of around 10,000 transitions. For each task, 10 randomly sampled frames are hand-annotated

**Figure 3.5: Evaluation Tasks.** Four Meta-World (top) simulation tasks and four real-robot tabletop manipulation tasks (bottom).

with saliency. Even with real-world images, only a small number of annotated frames are required to learn a good saliency predictor. We train an imitation learning policy by minimizing the mean-squared error between predicted end-effector pose and expert actions. We use a recurrent policy to encode history information and a 2-layer MLP to predict continuous actions.

ViSaRL scales to real-robot tasks and is robust to distractor objects. Videos of evaluation trajectories for each task can be found on the project website. Table 3.7 reports the task success rates on 10 evaluation rollouts; Table 3.6 summarizes the same tasks under human evaluation.

Table 3.7: Task success rates in real-world tabletop manipulation tasks for **RGB** and **RGB+Saliency** with MultiMAE.

Method	Apple	Red Block	Bread → Plate	Apple → Bowl	Cumulative
RGB	6/10	4/10	3/10	1/10	14/40
+Saliency	8/10	7/10	6/10	6/10	27/40

Even on the easier Pick Apple task, using saliency augmented representations, **RGB+Saliency**, improves the success rate over **RGB**. On tasks with distractors and longer-horizon tasks such as Put Apple in Bowl, saliency-augmented representations nearly double the success rate.

3.7 Conclusion

We proposed to use human-annotated saliency as an additional input modality for solving challenging visual robot control tasks. We present a simple approach, ViSaRL, to utilize saliency to learn robust image representations enabling more sample-efficient and generalizable policy learning.

Limitations and Future Work. One potential limitation of our user interface is that it could be tedious to collect saliency annotations when scaling to more complex real world applications or video saliency [296]. Future work could investigate alternative interfaces that will enable collecting more saliency data, e.g., area-based methods or by tracking the eye gaze of the user [236].

One can further evaluate the generalizability of ViSaRL on the recent benchmark, The Colosseum [240], a suite of manipulation tasks design to measure the robustness of trained robot policies against visual perturbations.

In this chapter, we only considered static frame saliency maps for single-object manipulation tasks. We plan to extend our approach to handle longer-horizon multi-object tasks using video saliency models [252] which can learn to encode more flexible temporal saliency representations across a sequence of frames. This extension could be implemented by asking the human users to watch video clips of the trajectories and annotate saliency over these clips.

CHAPTER 4

Chapter 4

HAND Me the Data:

Fast Robot Adaptation via Hand Path Retrieval

4.1 Introduction

For robots to operate seamlessly in human-centric settings, they should be able to *rapidly* learn new tasks with *minimal human supervision*. This requires learning algorithms that (1) scale across many tasks and (2) adapt quickly to new ones.

Imitation learning has produced capable multi-task robot policies [33, 155, 185, 230, 278], but scaling is hindered by its reliance on vast amounts of expert, task-specific teleoperation data [192]. In contrast, *task-agnostic play data* is far easier to collect, without requiring constant environment resets or task-specific labeling [207, 214, 322]. The challenge is in making such unstructured data usable for teaching robots new tasks quickly.

Therefore, we propose HAND, a *simple and time-efficient* approach to adapt pre-trained play policies to specific tasks using a single human hand demonstration (see Figure 4.1). Unlike prior retrieval methods [75, 194, 215, 226, 267, 301] that require robot demonstrations of the target task, HAND extracts 2D relative hand motion paths from the provided human hand demonstration to

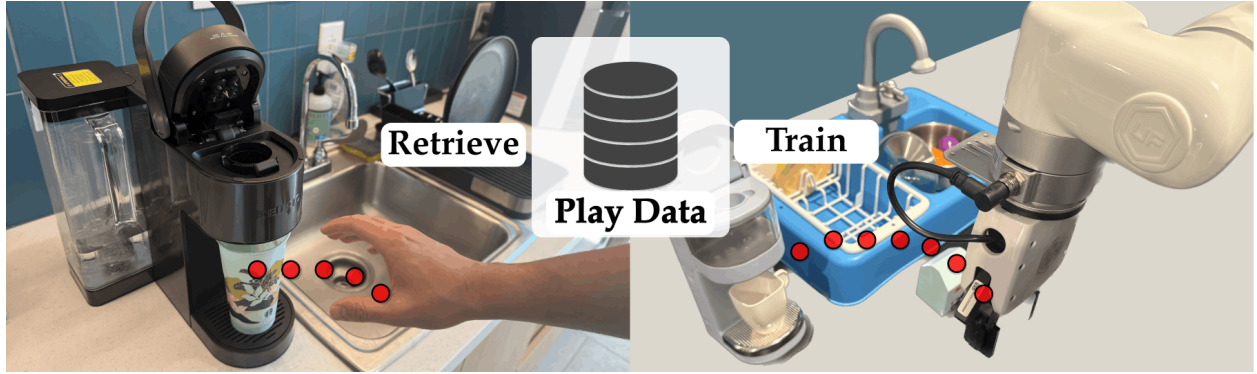


Figure 4.1: HAND learns a policy from as little as one (1) human hand demonstration.

inform retrieval, enabling even non-experts to teach robots without teleoperation.

HAND enables both *scalability* and *speed*. Towards *scalability*, HAND avoids the need for calibrated depth cameras [107, 237], specialized eye-in-hand setups [154], or detailed hand-pose estimation [154, 177]. Instead, it labels a robot play dataset with 2D gripper positions relative to the RGB camera frame, tracked using a visual point-tracking model [145, 145]. When a human hand demonstration is provided, HAND tracks the hand trajectory with the same simple pipeline. The hand positions are then converted into 2D *relative* sub-trajectories, capturing motion agnostic to the starting point [331]. After an initial filtering step that removes unrelated behaviors using a visual foundation model [233], HAND retrieves matching sub-trajectories from the play dataset based on the 2D relative hand path. Finally, towards *speed*, a policy pre-trained on the play dataset is LoRA-fine-tuned on the retrieved sub-trajectories, encouraging the policy to specialize in the demonstrated task. Because HAND retrieves primarily based on hand motion, it is robust to irrelevant visual features such as background clutter and lighting changes compared to purely visual retrieval methods.

Our experiments, across **10 tasks** and **550 total evaluations** in the real world on a WidowX robot demonstrate that HAND enables quick adaptation even to long-horizon tasks, outperforming the best baseline by 3× in task completion. We also demonstrate that HAND is effective with hand demonstrations collected from *completely different scenes* from the robot scene and across significant camera angle changes. Finally, we perform a *real-time learning* experiment, where

HAND learns a challenging long-horizon task in **under 4 minutes** of experiment time, from providing the hand demonstration to the trained policy, while being on average $5\times$ faster to collect data for than robot teleoperation demonstrations on our WidowX arm.

4.2 Related Works

Robot Data Retrieval. Prior work has demonstrated *retrieval* as an effective mechanism for extracting relevant on-robot data for training robots [75, 148, 194, 215, 226, 267, 268, 301]. For example, SAILOR [226] and Behavior Retrieval [75] pre-train variational auto-encoders (VAEs) on prior robot images and actions to learn a latent embedding. This latent embedding is used to retrieve states and actions from an offline dataset similar to ones provided in expert demonstration trajectories. However, retrieving based on learned full image encodings or even raw pixel values [267] can be noisy; Flow-Retrieval [194] instead trains a VAE to encode *optical flows* indicating movement of objects and the robot arm in the scene. Similar to Flow-Retrieval, HAND also retrieves based on robot arm motion. However, rather than training a dataset-specific VAE model that may not be robust to large visual differences, we retrieve from our offline robot data by primarily matching motions of a human hand demonstration using *relative 2D paths* of the robot end-effector in the prior data. This hand path retrieval helps us robustly retrieve relevant robot arm *behaviors*.

STRAP [215] addresses visual retrieval robustness issues of prior work by using features from DINO-v2 [233], a large pre-trained image-input foundation model for retrieval. However, STRAP, along with all aforementioned retrieval work, assumes access to expert robot demonstrations for the target tasks. In contrast, HAND only requires a *single*, easier-to-collect human hand demonstration that results in more *time-efficient* learning of demonstrated tasks compared to methods requiring robot teleoperation data. Furthermore, our experiments demonstrate that HAND retrieves more task-relevant trajectories and therefore attains higher success rates compared to these methods.

Learning From Human Hands. Similar to HAND, a separate line of work proposes using

human hands to learn robot policies. One approach is to train models on human video datasets to predict future object flows [311, 328] or human affordances [13, 163]. These intermediate affordance and flow representations are then used to either train a policy conditioned on this representation [311] on robot data or control a heuristic policy [13, 163, 328]. Other works focus on learning directly from human hands [107, 147, 154, 177, 237]. These works generally use hand-pose detection models aided by multiple cameras or calibrated depth cameras to convert hand poses directly to robot gripper keypoints [107, 177, 237]. However, works that exclusively retrieve human data are restricted to constrained policy representations as they must match human hand poses to robot gripper poses. Kim et al. [154] instead use an eye-in-hand camera mounted on a human demonstrator’s forearm to train an imitation learning policy conditioned on robot eye-in-hand camera observations. Unlike these prior works, HAND only requires a single RGB camera from which the robot gripper can be seen. Also, we focus on retrieving robot play data, allowing us to train arbitrarily expressive policies without constrained policy representations [107, 177, 237] or intermediate representations [13, 163, 311, 328].

4.3 HAND: Fast Robot Adaptation via Hand Path Retrieval

4.3.1 Preliminaries and Formulation

We assume access to a dataset of task-agnostic robot play data, $\mathcal{D}_{\text{play}}$, consisting of trajectories $\tau_i = \{(o_t, a_t)\}_{t=1}^T$, where each o_t is a per-timestep observation that includes RGB images of the robot gripper and robot proprioceptive information, and a_t is the robot action. These trajectories may span many scenes, tasks, and episode horizons. We do not assume task labels (e.g., language labels), as data collection is easier to scale without labeling each sub-trajectory in a long-horizon play trajectory.¹

In contrast to retrieval methods that rely on robot demonstrations for each target task [75, 194, 215, 226], we assume access to easy-to-provide human hand demonstrations. For each task, a human records their hand movement without teleoperating the robot. On our real-world

¹Section 4.4 demonstrates that HAND can also incorporate language labels as extra policy conditioning.

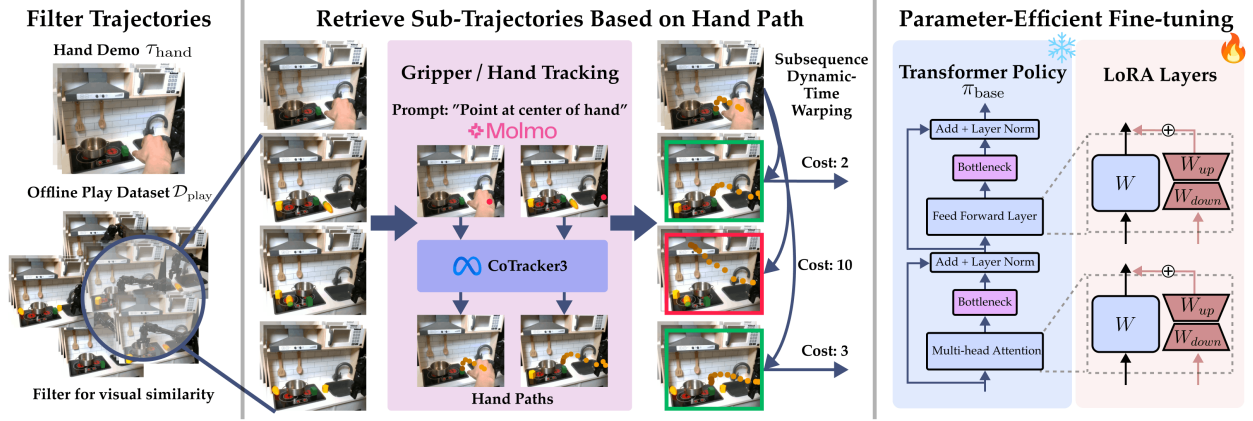


Figure 4.2: HAND enables fast-adaptation to a new target task by using an easy-to-provide hand demonstration of the target task (Left). We propose a two-step retrieval procedure where we first filter the trajectories in the offline play dataset, $\mathcal{D}_{\text{play}}$, for visually similar trajectories based on features from a pretrained vision model. We use off-the-shelf, pretrained hand detection and point tracking to construct 2D paths of the motion for both the human hand and robot end-effector. We use these paths as a distance metric to retrieve relevant trajectories from the play dataset (Middle) for quickly fine-tuning a pretrained transformer policy on the target task (Right).

setup, these hand demonstrations, $\mathcal{D}_{\text{hand}}$, are on average $5\times$ faster to collect than robot teleoperation data. Moreover, hand demonstrations are generally easier to provide than robot teleoperation [180, 304]. Each video in $\mathcal{D}_{\text{hand}}$ consists of RGB frames $o_1, \dots, o_H$, captured such that the human hand occupies a similar viewpoint in the frame as the robot gripper does in $\mathcal{D}_{\text{play}}$.²

Given $\mathcal{D}_{\text{play}}$ and $\mathcal{D}_{\text{hand}}$, we aim to train a policy $\pi_{\theta}(a | o)$ to perform the target task demonstrated by the human in $\mathcal{D}_{\text{hand}}$. Since we do not assume task labels in $\mathcal{D}_{\text{play}}$ and we are provided no expert robot teleoperation demonstrations, we must *retrieve* sub-trajectories indicating how to perform the behavior demonstrated in $\mathcal{D}_{\text{hand}}$ from $\mathcal{D}_{\text{play}}$ for training π . We denote this retrieved dataset, later used for imitation learning, as $\mathcal{D}_{\text{retrieved}}$. Moreover, following our motivation in Section 4.1, we aim for our method to be *fast*, so that non-expert end-users can easily train the robot for many downstream tasks.

The key challenges HAND addresses are: (1) designing a representation that can unify the behaviors in robot sub-trajectories and human hand demonstrations (Section 4.3.2), (2) retrieving relevant sub-trajectories based on a suitable distance metric between these representations

²Section 4.4 demonstrates HAND works under large camera angle shifts.

(Section 4.3.3), and (3) quickly training a policy that can perform various unseen target tasks with a high success rate without expert demonstrations (Section 4.3.4). See Figure 4.2 for an overview. See Figure 4.2 for an overview and Algorithm 9 for full algorithm pseudocode.

4.3.2 Path Distance as a Unifying Representation for Retrieval

Prior robot retrieval methods assume access to expert demonstrations from which they extract proprioceptive information (e.g., joint angles and actions) alongside visual features for retrieval [75, 194, 215, 226, 267]. However, since $\mathcal{D}_{\text{hand}}$ contains only visual data and no robot actions, retrieval based purely on appearance can be noisy—especially due to the visual domain gap between hand demonstrations in $\mathcal{D}_{\text{hand}}$ and robot demonstrations in $\mathcal{D}_{\text{play}}$ (see Figure 4.2, left). To address these issues, we propose an embodiment-agnostic, behavior-centric retrieval metric that enables matching between $\mathcal{D}_{\text{hand}}$ and $\mathcal{D}_{\text{play}}$ based on demonstrated behaviors rather than appearance.

Using 2D Paths for Retrieval. The movement of the robot end-effector over time provides rich information about its behavior [185]. We represent behaviors in both datasets using the paths traced by the human hand or the gripper. Because we assume access only to an RGB camera from which the hand or the gripper is visible (i.e., no depth), we construct these paths in 2D relative to the camera viewpoint for both $\mathcal{D}_{\text{play}}$ and $\mathcal{D}_{\text{hand}}$.³

Obtaining Paths from Data. To extract paths, we use CoTracker3 [145], an off-the-shelf point tracker capable of tracking 2D points across video sequences, even under occlusion. CoTracker3 only requires a single point on the gripper or hand to track motion across the full sequence. We use Molmo-7B [71], an open-source 7B image-to-point foundation model, to automatically select this point by prompting it at the *midpoint* of each trajectory with either “Point at the center of the hand” or “Point to the robot gripper.” Using the middle frame ensures a higher chance of visibility in case the gripper or hand is not yet in frame at the beginning or occluded at the end.⁴

Given the 2D point $(x, y)_{\text{hand}}$ or $(x, y)_{\text{play}}$ from the middle frame, we use CoTracker3 to perform

³If both datasets have additional calibrated depth information, HAND can also operate on 3D paths.

⁴Points can also be obtained heuristically, e.g., if the robot starts from the same position in each $\mathcal{D}_{\text{play}}$ traj.

bidirectional point tracking, resulting in a 2D path $p_{\text{hand}} = \{(x_t, y_t)_{\text{hand}}\}_{t=1}^H$ or $p_{\text{play}} = \{(x_t, y_t)_{\text{play}}\}_{t=1}^T$ for each trajectory. See the [Gripper/Hand Tracking](#) block of [Figure 4.2](#) for a visualization of this pipeline. Next, we describe how we use 2D paths to retrieve sub-trajectories from $\mathcal{D}_{\text{play}}$.

4.3.3 Retrieving Relevant Sub-Trajectories using Path Distance

Background. For identifying relevant sub-trajectories in $\mathcal{D}_{\text{play}}$, we use Subsequence Dynamic Time Warping (S-DTW) [219], an algorithm for aligning a shorter sequence to a portion of a longer reference sequence prior work has demonstrated effective for sub-trajectory retrieval [215]. Given a query sequence $Q = \{q_1, q_2, \dots, q_H\}$ and a longer reference sequence $R = \{r_1, r_2, \dots, r_T\}$, where $T > H$, the goal of S-DTW is to find a contiguous subsequence of R that minimizes the total cumulative distance between elements of both sequences. In HAND, the query sequences are the 2D hand demonstration paths $\{(x_t, y_t)_{\text{hand}}\}_{t=1}^H$ and the reference sequences are the 2D paths generated from long-horizon robot play data $\{(x_t, y_t)_{\text{play}}\}_{t=1}^T$.

Sub-Trajectory Preprocessing. To preprocess the datasets for S-DTW, we first segment the offline play dataset, $\mathcal{D}_{\text{play}}$, into variable-length sub-trajectories using a simple heuristic based on proprioception proposed in several prior works [215, 264]. In particular, we split the trajectories whenever the acceleration or velocity magnitude (depending on what proprioception data is available) drops below a predefined ϵ value, corresponding to when the teleoperator switches between tasks. We find that this simple heuristic can reasonably segment trajectories into atomic components resembling lower-level primitives. We also split the hand demonstrations evenly into smaller sub-trajectories based on how many subtasks the human operator determined they have completed. After sub-trajectory splitting, we have two sub-trajectory datasets, $\mathcal{T}_{\text{hand}} = \{t_{1:a}^i, t_{a:b}^i, \dots, t_{H_i-|p_{\text{hand}}^i|:H_i}^i \mid \forall \tau_{\text{hand}}^i \in \mathcal{D}_{\text{hand}}\}$ and $\mathcal{T}_{\text{play}} = \{t_{1:a}^j, t_{a:b}^j, \dots, t_{T_j-|p_{\text{play}}^j|:T_j}^j \mid \forall \tau_{\text{play}}^j \in \mathcal{D}_{\text{play}}\}$ where $|p_{\text{hand}}^i|$ and $|p_{\text{play}}^j|$ are the lengths of the last sub-trajectory paths of trajectories i, j from $\mathcal{D}_{\text{hand}}$ and $\mathcal{D}_{\text{play}}$, respectively. Finally, each sub-trajectory is represented in *relative 2D coordinates*, i.e., $p_t = [x_{t+1} - x_t, y_{t+1} - y_t]$. Relative coordinates ensure retrieval invariance to the initial positions of the hand or gripper [331].

Visual Filtering. A key limitation of path distance retrieval is that distinct tasks can exhibit similar movement patterns. For example, tasks like “pick up the mug” and “pick up the cube” can appear nearly identical in 2D path space [185]. But, the retrieved trajectories for one task may not benefit learning of the other; since we do not assume task labels in $\mathcal{D}_{\text{play}}$, a policy directly trained on “pick up the cube” retrieved sub-trajectories may still fail to pick up a mug. Therefore, before retrieving sub-trajectories with paths, we first run a visual filtering step to ensure that the sub-trajectories we retrieve will be task-relevant. We use an object-centric visual foundation model, namely DINOv2 [233], to first filter out sub-trajectories performing unrelated tasks with different objects. Specifically, we use the DINOv2 first and final frame embedding differences, representing visual object movement from the first to last frame, between human hand demonstrations and robot play data to filter $\mathcal{T}_{\text{play}}$. In practice, this simple filtering step removes the majority of irrelevant sub-trajectories. For a given image sequence $o_{1:H}^{\text{hand}}$ from a hand sub-trajectory and image sequence $o_{1:T}^{\text{play}}$ from a robot play sub-trajectory, we define the cost as:

$$C_{\text{visual}}(o_{1:H}^{\text{hand}}, o_{1:T}^{\text{play}}) = \underbrace{\| \text{DINO}(o_1^{\text{hand}}) - \text{DINO}(o_1^{\text{play}}) \|_2^2}_{\text{first-frame DINO}} + \underbrace{\| \text{DINO}(o_H^{\text{hand}}) - \text{DINO}(o_T^{\text{play}}) \|_2^2}_{\text{last-frame DINO}}. \quad (4.3.1)$$

We take the M trajectories with lowest cost as possible retrieval trajectories from $\mathcal{D}_{\text{play}}$ for each human demonstration sub-trajectory in $\mathcal{T}_{\text{hand}}$. The rest are ignored for those hand demonstrations.

Retrieving Sub-Trajectories. We then employ S-DTW to match the target sub-trajectories, $\mathcal{T}_{\text{hand}}$, to the set of visually filtered segments $\in \mathcal{T}_{\text{play}}$. Given two sub-trajectories, $t_i \in \mathcal{T}_{\text{play}}$ and $t_j \in \mathcal{T}_{\text{hand}}$, S-DTW returns the cost along with the start and end indices of the subsequence in t_j that minimizes the path cost (see Figure 4.2). We select the K matches from $\mathcal{D}_{\text{play}}$ with the lowest cost to construct our retrieval dataset, $\mathcal{D}_{\text{retrieved}}$.

4.3.4 Putting it All Together: Fast-Adaptation with Parameter-Efficient Policy Fine-tuning

We aim to enable fast, data-efficient learning of the task demonstrated in $\mathcal{D}_{\text{hand}}$. To this end, we first pretrain a task-agnostic base policy π_{base} on $\mathcal{D}_{\text{play}}$ with standard behavior cloning (BC) loss. While our approach is compatible with any policy architecture, we use action-chunked

transformer policies [338] due to their suitability for low-parameter fine-tuning and strong performance in long-horizon imitation learning [33, 109, 338, 339].

Adapting to $\mathcal{D}_{\text{retrieved}}$. To rapidly adapt to a task with minimal data, we leverage parameter-efficient fine-tuning using *task-specific adapters*—small trainable modules that modulate the behavior of the frozen base policy. Adapter-based methods have shown promise in few-shot imitation learning [186, 202], making them ideal for our limited retrieved dataset $\mathcal{D}_{\text{retrieved}}$. Specifically, we insert LoRA layers [122] into the transformer blocks of π_{base} . These are low-rank trainable matrices (about 0.1%–2% of the parameters of π_{base}) added to the attention projections of π_{base} (see Figure 4.2, **LoRA Layers**). During fine-tuning, we update only the parameters of these LoRA layers, θ , using $\mathcal{D}_{\text{retrieved}}$.

Loss Reweighting. While our retrieval mechanism identifies sub-trajectories relevant to the target task, not all will be equally useful. Following prior work [267, 268, 301], we reweight the BC loss with an exponential term $\in (0, \infty)$ (similar to AWR [238]), where each sub-trajectory is weighted based on its S-DTW similarity to the hand demonstration. Intuitively, this upweights the loss of the most relevant examples in $\mathcal{D}_{\text{retrieved}}$ and downweights those that are less relevant. Finally, because trajectory cost scales vary depending on the task being retrieved and the features being used for S-DTW, we rescale the S-DTW costs $C_{i,\text{path}}$ to a fixed range. For each $\tau_i \in \mathcal{D}_{\text{retrieved}}$, its weight $e^{-C_{i,\text{path}}}$ is scaled to between $[0.01, 100]$, where the normalization term comes from the sum of costs of all trajectories in $\mathcal{D}_{\text{retrieved}}$. Let the normalized weight for a trajectory be $w_i = \exp(-C_{i,\text{path}})$ and the behavioral cloning loss be $L_i(a, o) = -\log \pi_\theta(a \mid o)$. The total loss is then the weighted average over the dataset $\mathcal{D}_{\text{retrieved}}$:

$$\mathcal{L}_{\text{BC};\theta} = \frac{1}{|\mathcal{D}_{\text{retrieved}}|} \sum_{\tau_i \in \mathcal{D}_{\text{retrieved}}} w_i \times L_i(a, o). \quad (4.3.2)$$

We summarize HAND in the pseudocode in Algorithm 2. For the full algorithm, see Algorithm 9; for full implementation details and hyperparameters, see Section B.1.

Algorithm 2 HAND PSEUDOCODE

Require: $\mathcal{D}_{\text{hand}}$, $\mathcal{D}_{\text{play}}$, threshold ϵ , # visual-filtered trajectories M , # retrieved sub-trajectories K

- 1: Train base policy π_{base} on $\mathcal{D}_{\text{play}}$ via behavior cloning
- 2: Segment both $\mathcal{D}_{\text{play}}$ and $\mathcal{D}_{\text{hand}}$ into sub-trajectory datasets w/ threshold ϵ : $\mathcal{T}_{\text{play}}$, $\mathcal{T}_{\text{hand}}$
- 3: **for** $\tau^{\text{hand}} \in \mathcal{T}_{\text{hand}}$ **do**
- 4: Filter top- M visually similar $\tau^{\text{play}} \in \mathcal{T}_{\text{play}}$ via DINO-based C_{visual}
- 5: **for** each filtered τ^{play} **do**
- 6: Track 2D hand paths with Molmo + CoTracker3
- 7: Retrieve K best-matching segments via S-DTW on relative path similarity
- 8: Fine-tune π_{base} on retrieved data with adapter layers θ to obtain π_{θ} with $\mathcal{L}_{\text{BC};\theta}$ Equation (4.3.2)
- 9: **return** π_{θ}

4.4 Experiments

We evaluate HAND both as a retrieval pipeline and as a method for quickly learning downstream tasks. To this end, we organize our experiments to answer the following questions:

- (Q1) How well can HAND retrieve *task-relevant* behaviors?
- (Q2) Does HAND support hand demonstrations from *unseen scenes* and is it *robust* to visual shifts?
- (Q3) How does HAND perform in policy learning?
- (Q4) Can HAND enable *real-time* adaptation?

4.4.1 Experimental Setup

We evaluate HAND on a real-world multi-task kitchen environment using the WidowX robot arm. Our robot environment setup is shown in Figure 4.3. We use an Intel Realsense D435 camera as an external camera and a Logitech C920 as an over-the-shoulder camera.

Evaluation Tasks: We evaluate on 10 total tasks: three standard tasks—REACH GREEN BLOCK, PRESS



Figure 4.3: WidowX Robot Arm Setup. We evaluate the scalability of HAND on 10 manipulation tasks on a WidowX robot arm in a kitchen setup [292].

BUTTON, and CLOSE MICROWAVE—and three challenging long-horizon tasks—PUT K-CUP IN COFFEE MACHINE, BLEND CARROT, and COOK CARROT. The latter tasks demand high precision and span more than 150 timesteps at a 5 Hz control frequency. In particular, COOK CARROT is composed of four shorter tasks, SLIDE POT → PUT OBJECT IN POT → PUT LID ON POT → TURN STOVE KNOB, including non-prehensile tasks (e.g., slide pot) and taking ~ 300 steps to complete even for expert teleoperators. For our long-horizon tasks, we provide one hand demonstration to perform retrieval for each subtask. Partial success is provided for tasks composed of multiple subtasks.

Play Dataset Collection: We collect a task-agnostic play dataset containing a total of 50k transitions, each trajectory having an average of 230 timesteps and covering multiple tasks, collected at 5 Hz. The full dataset required roughly four hours to collect. To prevent the play data from mirroring evaluation tasks, we place distractor objects in the environment for teleoperators to interact with during collection. The dataset is split into two, with about 1 hour corresponding to the scene for COOK CARROT. To evaluate language-conditioned methods, we manually annotate the COOK CARROT scene of the dataset with language, which takes an additional 87 minutes. During data collection and evaluation, movable task objects are randomized in a 5" x 7" region within the workspace.

Baselines: We compare **HAND** to the following baselines:

- π_{base} : the base policy pre-trained on play data;
- LCBC: π_{base} with language-conditioning;
- **CLIP-L**: retrieves based on cosine similarity between language embedding of target task (rather than hand demo) and language embedding of the play data;
- **CLIP-I**: retrieves based on cosine similarity between language embedding of target task and image embedding of the play data;
- **Flow** [194]: trains a VAE on pre-computed optical flows for $\mathcal{D}_{\text{play}}$ from GMFlow [310] and retrieves individual states-action pairs based on latent motion similarity; and
- **STRAP** [215]: also uses S-DTW for sub-trajectory retrieval but uses S-DTW distance based solely on Euclidean distance between pre-trained DINO-v2 image embeddings.

Table 4.1: Number of retrieved sub-trajectories performing the demonstrated task. **HAND** retrieves more task-performing sub-trajectories than **Flow** and **STRAP**.

	Reach Green Block	Push Button	Close Microwave
Flow	7/25	0/25	0/25
STRAP	5/25	0/25	2/25
HAND(-VF)	9/25	13/25	9/25
HAND	15/25	18/25	11/25

STRAP and **Flow** assume access to expert *robot* demonstrations for both retrieval and fine-tuning. In our setting, we do not assume such demonstrations, therefore, unless otherwise noted, we adopt them without expert fine-tuning. While **STRAP** and **Flow** originally propose training policies from scratch, we instead apply LoRA fine-tuning—as with **HAND**—which we found to yield better performance for these baselines.

Policy Architecture: To ensure fair comparison, all methods use a three-layer action-chunking transformer (similar to ACT [338]) decoder policy where applicable. The input to the transformer policy is a sequence of image tokens corresponding to the external and over-shoulder camera views. Conditioned on the current image observation, the model predicts an action chunk corresponding to a second of execution. We refer the reader to Section B.1 for implementation details and Section B.5 for extensive ablation results.

4.4.2 Experimental Evaluation

(Q1): HAND retrieves more task-relevant data. We analyze the quality of retrieved sub-trajectories between **Flow**, **STRAP**, and **HAND**. **STRAP** and **HAND** both use S-DTW-based trajectory retrieval, but **STRAP** relies purely on visual DINO-v2 embeddings for retrieval. We provide a single hand demonstration of three real robot tasks and retrieve the top $K = 25$ matches from $\mathcal{D}_{\text{play}}$. As shown in Table 4.1, **HAND** retrieves more task-relevant trajectories than both **STRAP** and **Flow**. **STRAP** relies exclusively on visual similarity, while **Flow** relies exclusively on motion similarity. Both methods struggle to bridge the domain gap between human hand demonstrations and robot

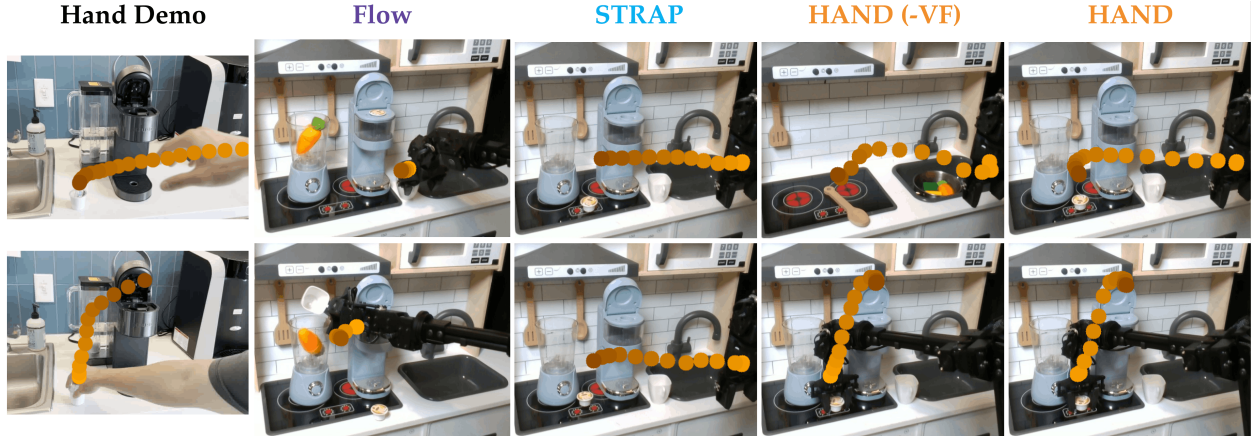


Figure 4.4: Qualitative retrieval results on OOD scene. We visualize the top sub-trajectory match of **Flow**, **STRAP**, **HAND (-VF)**, and **HAND** on two OOD domain demonstrations recorded from an iPhone camera, showing approaching a K-Cup and putting it into the machine. Only **HAND**’s top match is relevant for both hand demonstrations.

Table 4.2: Camera angle robustness results. Number of relevant retrieved trajectories for PUT LID ON POT if we change the camera angle vertically and horizontally by 10° increments. **HAND** retrieves +18% more relevant trajectories compared to **STRAP** even in the extreme case of 30° shift.

Method	10° Horiz.	20° Horiz.	30° Horiz.	10° Vert	20° Vert	30° Vert
Flow	0 / 25	2 / 25	5 / 25	1 / 25	0 / 25	0 / 25
STRAP	1 / 25	10 / 25	13 / 25	12 / 25	11 / 25	11 / 25
HAND	21 / 25	18 / 25	19 / 25	16 / 25	13 / 25	14 / 25

play data. In particular, for PUSH BUTTON, **STRAP** is unable to retrieve any relevant trajectories in its top matches.

We also observe that **visual filtering is necessary** to retrieve trajectories where the target object is interacted with, as demonstrated by **HAND (-VF)**, an ablation of **HAND** without visual filtering (Section 4.3.3), having 30% worse retrieval performance than **HAND** in Table 4.1.

(Q2): HAND supports hand demonstrations from unseen environments and is robust to camera angle shifts. Because **HAND** retrieves based on *relative hand motions*, it is also effective with hand demonstrations from out-of-distribution (OOD) scenes. To illustrate, we collect hand demonstrations in a new environment using a handheld iPhone camera and a real coffee machine, while retrieving from robot play data recorded in a completely different scene with a

Table 4.3: Long horizon COOK CARROT task results. We show success rates on each subtask and on the full task execution. Successes are out of 10.

Method	Slide Pot	Put Obj. in Pot	Put Lid on Pot	Turn Knob	Long Horiz.
LCBC	2	0	0	0	0
CLIP-L	0	4	1	0	0
STRAP	0	0	2	0	0
HAND w/o Pre-training	1	1	1	2	0
HAND (-VF)	2	0	0	5	0
HAND (-CW)	2	5	3	5	0
HAND	5	6	4	6	3
HAND + LCBC	8	7	5	7	5

toy coffee machine. In Figure 4.4, we show the lowest cost retrieved sub-trajectory of STRAP and Flow compared to HAND and a HAND ablation without the visual filtering step, HAND (-VF). Both of the retrieved trajectories for STRAP and Flow, along with the top trajectory for HAND (-VF) are irrelevant to the demonstrated task. For the first task, STRAP is able to retrieve the initial reaching motion toward the K-Cup but misses the crucial grasping segment, as it does not leverage motion for retrieval. Only HAND retrieves relevant robot trajectories for both hand demonstrations because it focuses on the *motion* demonstrated by the human hand after *visual filtering*.

Table 4.2 shows that HAND is also robust to shifts in camera angle for the PUT LID ON POT task, far more than Flow and STRAP. We measure the number of relevant retrieved trajectories of different methods after vertical and horizontal camera angle shifts of 10° increments. In the most extreme setting of 30° shift, HAND retrieves +18% more relevant trajectories compared to STRAP. These camera angle shifts emulate head rotations on humanoid robots or camera movement on mobile manipulators, suggesting that HAND can even work in such settings where camera viewpoint change may occur.

(Q3): HAND enables efficient policy learning in the real world. We evaluate four methods, including ours, across three standard tasks with ten trials each, for a total of **120 evaluations**. Real-world experiments in Figure 4.6 on our three standard tasks demonstrate that fine-tuning



Figure 4.5: Fast Adaptation Study. We conduct a small-scale user study to demonstrate HAND’s ability to learn robot policies in real-time. From providing the hand demonstration (Left), to retrieval and fine-tuning a base policy (Middle), to evaluating the policy (Right), we show HAND can learn to put a carrot in the blender with 7.5/10 task completion in less than 4 minutes.

with HAND improves success rates by +45% over the next best baseline, STRAP. In contrast, Flow fails to learn a policy that achieves reasonable success rates in any of the tasks. We also report the performance of π_{base} , trained on all of $\mathcal{D}_{\text{play}}$ and note that the pre-trained policy struggles to perform the tasks without any task-specific fine-tuning.

We next evaluate eight methods, including several ablations of HAND, across four base tasks and one long-horizon task constructed from these base tasks, with ten trials each for a total of **400 evaluations**. Results on the more challenging long-horizon tasks (Table 4.3) demonstrate that retrieval using **hand demonstrations outperforms language-based retrieval (CLIP-L)** by a fac-

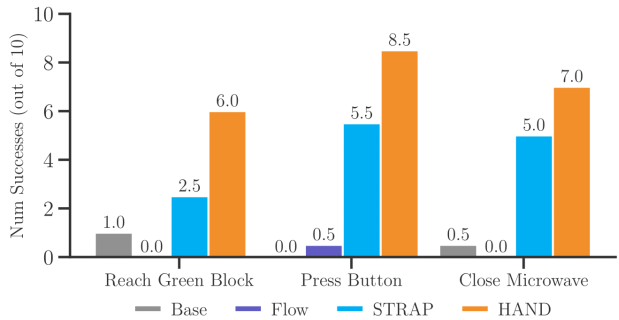


Figure 4.6: Real Robot Results. Task completion (including partial completion) out of 10 of π_{base} , STRAP, Flow, and HAND.

tor of 3× in success rate. Language-based retrieval suffers from the lack of spatial awareness, often retrieving trajectories that are semantically correct but spatially misaligned—similar to STRAP—which makes policy fine-tuning more difficult. In contrast, **directly conditioning on language performs poorly** compared to retrieval (LCBC), despite the fact that annotating sub-trajectories with language more than doubles data collection and annotation time.

Ablation Study: We observe that each component of HAND, namely pretraining, visual fil-

Table 4.4: Hand vs. Robot Teleoperation. **HAND** uses a single hand demonstration while **STRAP** uses robot teleoperation demonstrations. **HAND** achieves higher success rates in significantly less data collection time.

Method	Time (Min) ↓		Success Rate ↑	
	User 1	User 2	User 1	User 2
HAND (Hand Demo)	3	2	5/10	4/10
STRAP (Robot Demo)	10	14	3/10	2.5/10

tering (VF), and cost weighting (CW), are critical for task performance. Cost weighting helps bias the resulting policy towards behaviors that are most relevant to the downstream task, and reduces the effect of potentially noisy retrievals that may not directly aid in learning the target task. Without any of these components, the resulting policy is unable to learn the task. Only HAND completes COOK CARROT, succeeding in 3 out of 10 trials. We also show that given access to a language-annotated dataset, one could add language-conditioning on top of **HAND** to further improve the task performance (**HAND** + LCBC).

(Q4): HAND enables real-time, data-efficient policy learning of long-horizon tasks.

Finally, we performed two small-scale user studies with IRB approval from our institution to demonstrate real-time learning. In the first study shown in Figure 4.5, a participant familiar with HAND iteratively demonstrated each part of a long-horizon BLEND CARROT task and trained a HAND policy *with over 70% success rate in under four minutes* from providing a single hand demonstration to deploying the fine-tuned policy. A video of a similar experiment can be found on our website.

Hand vs Robot Demonstration Comparison: In the second study, two users with prior teleoperation experience—but not affiliated with this research—each collected a total of 20 demonstrations: 10 using hand demonstrations and 10 using robot teleoperation, to train the robot for PUT K-CUP IN COFFEE MACHINE. We employ **HAND** retrieval for hand-collected demonstrations and **STRAP** retrieval for robot teleoperation demonstrations. For a direct comparison, we additionally fine-tune **STRAP** with the human-collected teleoperated demonstrations as per [215].

As reported in Table 4.4, teleoperated demonstrations required over $3\times$ more time to collect than hand demonstrations. Remarkably, with just a single hand demonstration per user, we fine-tuned a policy achieving over 40% task completion compared to STRAP which reaches only 25% using a single *robot teleoperation* demonstration. Interestingly, we observed that increasing the number of expert demonstrations for STRAP degraded downstream performance likely due to lower quality retrieved trajectories. These results demonstrate that HAND enables *fast* adaptation to downstream tasks with as few as one easy-to-provide hand demonstration.

4.5 Conclusion and Limitations

We presented HAND, a simple and time-efficient framework for adapting robots to tasks using easy-to-provide human hand demonstrations. We demonstrated that HAND enables *real-time* task adaptation with a *single* hand demonstration in under four minutes.

Extending to 3D paths for retrieval. While HAND uses 2D paths for retrieval, one future direction could extend HAND to estimate the hand trajectory in 3D using foundation depth prediction models. Another direction future work could consider is a mixture of features for improving retrieval for tasks that require more dexterous control, i.e., cloth folding or deformable object manipulation.

Severe Camera Viewpoint Changes. Future work could address issues from severe camera viewpoint shifts between the collected hand demonstrations and robot play data via the use of 3D information, multiple camera viewpoints, or scene re-rendering with virtual cameras [99].

CHAPTER 5

Chapter 5

CLAM: Continuous Latent Action Models for Robot Learning from Unlabeled Demonstrations

5.1 Introduction

Recent efforts in robotics attempted to tackle the collection of large-scale robot datasets [33, 61, 308]. However, data collection ultimately requires manual teleoperation, which is expensive and time-consuming. Furthermore, fine-tuning models in situ via teleoperation to address the diversity of real-world environments and tasks is challenging for laypeople.

A promising alternative is training on human video demonstrations. These are easier to collect than teleoperation data [119] and thus easier to scale to large datasets [67, 100]. However, utilizing human videos to train robot policies has two main challenges. First, standard imitation learning requires demonstrations from the robot’s own embodiment and viewpoint. Recent work addresses this by converting human videos to robot-compatible observations using hand tracking and inpainting [176, 177]. However, even with converted video, imitation learning still requires ground-truth robot control signals, which these conversion methods do not provide. This is precisely the problem we focus on in this chapter: learning from expert demonstrations captured as

Stage 1: CLAM Pretraining

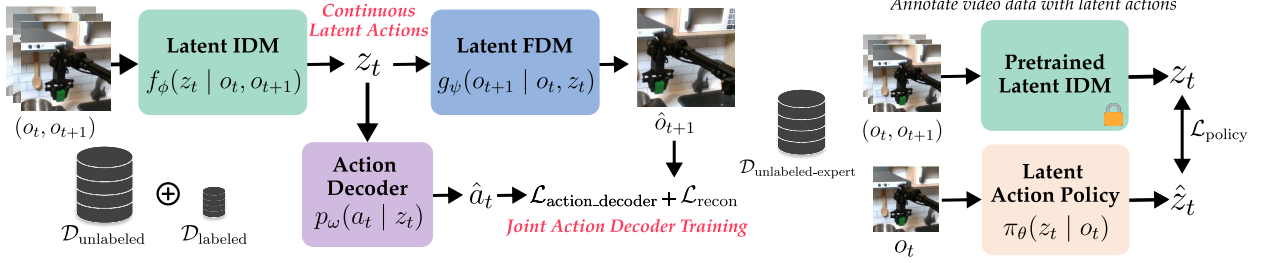


Figure 5.1: Overview of CLAM. CLAM consists of a **latent inverse dynamics model**, f_ϕ , which infers the latent action between consecutive observations and **latent forward dynamics model**, g_ψ , which predicts the future observation conditioned on the latent action and observation history. CLAM learns a latent action space through the self-supervised objective of future observation reconstruction. Unlike prior work, CLAM produces **continuous latent actions**. To ensure the learned latent space is amenable to decoding to real-world actions, CLAM **jointly** trains the action decoder and the latent action model.

robot video, but without action labels.

Actionless robot video is abundant in practice: human demonstrations can be converted to robot viewpoints [176, 177], internet video captures manipulation behaviors at scale [67, 100], and even existing robot datasets often contain observation logs without compatible action formats. What is missing is a method to recover executable actions from these observations.

We introduce *continuous latent action models* (CLAM), a framework for learning robot control policies from observation-only robot demonstrations. CLAM trains a latent inverse dynamics model to infer a continuous latent action between consecutive observations, supervised only by a forward dynamics model that must reconstruct the next observation from the current observation and the inferred latent action. This self-supervised loop requires no action labels and can be trained on large amounts of unlabeled video. To make these latent actions executable on a real robot, CLAM jointly trains an action decoder that maps latent actions to motor commands, using only a small amount of task-agnostic play data. Joint training is critical: it regularizes the latent space so that it remains decodable, which we show is necessary for continuous control. Once trained, the latent inverse dynamics model annotates unlabeled expert demonstrations with pseudo-actions, and a policy is trained via standard imitation learning on the resulting dataset.

However, realizing this pipeline for physical robots requires representations that map seamlessly to continuous, high-dimensional action spaces. In this work, we identify several key ar-

chitectural improvements that greatly improve the paradigm of learning from observation-only data in continuous control robotics settings. Our contributions are threefold:

1. We demonstrate that latent action relabeling becomes practical for high-dimensional continuous robot control when using continuous latent spaces and joint grounding.
2. We demonstrate that task-agnostic play data, rather than expert teleoperation, is sufficient for grounding latent actions, dramatically reducing the cost of data collection for new tasks.
3. We deploy and evaluate CLAM on a physical WidowX robot across four manipulation tasks, showing it is competitive with BC trained with privileged expert action labels without ever collecting action-labeled expert demonstrations.

5.2 Related Work

Imitation from Observation. Learning robot policies from sequences of observations without access to corresponding actions is often referred to as *imitation from observation* (IfO) [201]. Unlike standard imitation learning, the demonstrations may differ in embodiment, environment, or viewpoint, and observations do not include action labels. Existing IfO approaches either learn policies directly from observation sequences by aligning trajectories [198, 201, 314], or predict future observations and map them to robot controls using learned or off-the-shelf perception models [158, 299]. However, these methods often depend on online rollouts or off-the-shelf vision models, which limit their applicability. In contrast, CLAM is trained fully offline, without rollouts or off-the-shelf models.

Supervised Learning of Inverse Dynamics Models. With access to some action-labeled data, inverse dynamics models (IDMs) can be trained in a supervised fashion. VPT [17] and Seer [283], for example, use an IDM to label large unlabeled datasets, while UniPi [76] finetunes a video prediction model and applies an IDM to infer robot actions from synthetic data. A key limitation of such methods is that the IDM must predict actions in the *real* action space. Consequently, these methods are bottlenecked by the number of labeled demonstrations for task-specific finetuning or the budget required to collect them.

Latent Action Models. To bypass the dependency on labeled data, several works learn IDMs in a *latent* action space using only observation data [42, 43, 50, 65, 157, 216, 256, 319]. These works apply a variety of subtle architectural differences, which we compare in a real-world setup. Works such as LAPA [319] and DynaMo [65] use latent action learning for pretraining to achieve better representation learning performance [65, 281, 319]. However, unlike CLAM, since the latent action model (LAM) is limited to pretraining, each new task still requires collecting labeled expert data to train a policy. Works such as LAPO [256] and Genie [42] combine VPT-style methods with latent action models, using the IDM to annotate unlabeled trajectories directly. While promising, they focus on video game domains with small discrete action spaces. These methods use vector quantization in the latent action space; recent works point toward the benefits of using continuous latent action spaces [228, 316]. We perform a detailed analysis in the real-world robotics domain, validating key architectural choices that make latent action modeling practical in fully continuous, high-dimensional control settings.

A slew of concurrent works study the applicability of latent action learning to in-the-wild data [96], as a component of large behavior models [193], and world models [95, 320], highlighting the importance of this line of research.

Most similar to our work is the work by Nikulin et al. [228]. The authors find that supervision during LAM training and continuous action spaces are beneficial in simulated continuous control tasks. Our work differs due to our focus on real-world dexterous control and the use of off-domain data for large-scale pretraining, whereas Nikulin et al. [228] focus on invariance to distractors.

5.3 Problem Setting

We consider a practical setting in which a user wants to teach a robot a new task without collecting action-labeled expert demonstrations. The data regime involves three datasets with complementary roles.

The bulk of the training data comes from a large corpus of robot video collected across diverse environments and tasks, without action labels. This data may come from prior deployments,

passive recording, or, as recent work has shown, human videos converted to robot-viewpoint observations [176, 177]. We denote this pool $\mathcal{D}_{\text{unlabeled}}$. Because no teleoperation or action logging is required, this type of data is cheap to scale.

Separately, we assume access to a small dataset of action-labeled robot transitions, $\mathcal{D}_{\text{labeled}}$, collected task-agnostically. In our experiments, this consists of undirected play data, which requires minimal expertise from the operator and no task-specific planning, making it substantially cheaper to collect than expert demonstrations [207]. $\mathcal{D}_{\text{labeled}}$ is used solely to ground latent actions into executable motor commands.

The user’s only contribution is a small set of task-specific demonstrations, $\mathcal{D}_{\text{unlabeled-expert}}$, collected by performing the target task themselves. These demonstrations are expert with respect to the task but contain no action labels; the user simply shows what the task looks like, through a converted human video. The user’s demonstrations are used for two purposes. First, they are added to $\mathcal{D}_{\text{labeled}}$ for training the LAM and to ensure the LAM does not suffer out-of-distribution errors. Second, they are used as expert data to train the final policy.

To isolate the challenge of action inference from the orthogonal challenge of cross-embodiment transfer, we use robot demonstrations with action labels withheld as our unlabeled data. Additionally, all methods, including CLAM and baselines, have access to the same $\mathcal{D}_{\text{unlabeled}}$, $\mathcal{D}_{\text{labeled}}$, and $\mathcal{D}_{\text{unlabeled-expert}}$, however, some baselines might be bottlenecked by labeled demonstration requirements and thus cannot make use of some of the data. Furthermore, we focus on the single-task, single-robot-embodiment setting.

5.4 Continuous Latent Action Models

We introduce *continuous latent action models* (CLAM), shown in Figure 5.1, a scalable approach for training continuous control policies from unlabeled observation data. CLAM consists of two stages. In **Stage 1** (Section 5.4.1), we train a latent action model (LAM) for relabeling observation-only data. We then use this LAM in **Stage 2** (Section 5.4.2) to train a latent action policy.

Algorithm 3 CLAM w/ Joint Action Decoder Training

```
1: Input:  $\mathcal{D}_{\text{unlabeled}}, \mathcal{D}_{\text{labeled}}, \mathcal{D}_{\text{unlabeled\_expert}},$  IDM  $f_\phi$ , FDM  $g_\psi$ , Action Decoder  $p_\omega$ , Latent A. Policy  $\pi_\theta$   
    $N_C$ : number of CLAM update steps  
    $N_P$ : number of policy updates steps  
    $K$ : train action decoder every  
   # Stage 1: Train CLAM and Action Decoder  
2: for iter = 1 in range( $N_C$ ) do  
3:   Update  $f_\phi$  and  $g_\psi$  with  $\mathcal{L}_{\text{recon}}$  on  $\mathcal{D}_{\text{unlabeled}}$   
4:   if iter %  $K == 0$  then  
5:     Update  $p_\omega$  and  $f_\phi$  with  $\mathcal{L}_{\text{action\_decoder}}$  on  $\mathcal{D}_{\text{labeled}}$   
   # Stage 2: Train Latent Action Policy  
6: Annotate  $\mathcal{D}_{\text{unlabeled\_expert}}$  with IDM  $f_\phi$   
7: for iter = 1 in range( $N_P$ ) do  
8:   Update  $\pi_\theta$  with  $\mathcal{L}_\pi$  on annotated  $\mathcal{D}_{\text{unlabeled\_expert}}$ 
```

5.4.1 Latent Action Model Training

In **Stage 1**, we pretrain a Latent Action Model (LAM) that we later use to annotate trajectories with pseudo-action labels. A LAM consists of two components: a *forward dynamics model* (FDM), which predicts the transition dynamics of the environment, and an *inverse dynamics model* (IDM), which inverts this process by inferring the action performed between two subsequent observations.

Algorithm 4 Inference Time Rollout

```
1: Input: Action Decoder  $p_\omega$ , Latent Policy  $\pi_\theta$   
2:  $o_t = \text{env.reset}()$   
3: while not done do  
4:    $z_t = \pi_\theta(\cdot \mid o_t)$  // infer latent action  
5:    $a_t = p_\omega(\cdot \mid z_t)$  // decode latent action  
6:    $o_t, \text{done} = \text{env.step}(a_t)$ 
```

Since we train these models without any action labels, we train a **latent IDM**, $f_\phi(z_t \mid o_t, o_{t+1})$, which predicts an unobserved *latent* action z_t between two consecutive observations. To provide a training signal for the latent action, we jointly train a **latent FDM**, $g_\psi(o_{t+1} \mid o_t, z_t)$, to infer the next observation conditioned on the current observation and latent action. Since observations are *partial* and do not capture the full environment state, in practice, we provide the LAM with additional H steps of context making it easier to infer the underlying state and predict a more

accurate latent action, i.e., $f_{\phi}(z_t \mid o_{t-H}, \dots, o_t, o_{t+1})$ and $g_{\psi}(o_{t+1} \mid o_{t-H}, \dots, o_t, z_t)$.

As shown in Figure 5.1, the training signal comes from future observation reconstruction, i.e. $\mathcal{L}_{\text{recon}} = \text{MSE}(\hat{o}_{t+1}, o_{t+1})$ where $\hat{o}_{t+1}$ is the FDM prediction. Our encoder/decoder architecture induces an information bottleneck, ensuring a meaningful, compact action representation rather than shortcut solutions. While prior works [42, 256] discretize latent actions from the IDM using Vector Quantization (VQ) [287], our experiments show that this fails in robotics tasks where actions are inherently continuous. We solve this shortcoming by replacing the VQ-based discretized action space with a learned continuous actions.

Latent Action Decoder. At test time, the learned latent actions cannot be directly executed in the environment. Consequently, we learn a **latent action decoder**, $p_{\omega}(a_t \mid z_t)$ using $\mathcal{D}_{\text{labeled}}$ to ground the learned latent actions to executable environment actions. Some prior work [256] trains the action decoder independently from the latent action model. This is reasonable for environments with discrete action spaces, where it is possible to learn the mapping from a discrete set of codes to the environment actions. Recent work [228] points toward the benefit of providing additional supervision by jointly training with labeled data. We find this holds in real-world experiments, so we adopt a joint training approach for our LAM training. We find that this allows for effective decoding to real-world actions.

Importantly, we do not make any assumptions about how $\mathcal{D}_{\text{labeled}}$ is collected. Furthermore, we demonstrate that $\mathcal{D}_{\text{labeled}}$ *can come from any behavioral policy, even a random policy or task-agnostic play data*, allowing CLAM to work even without access to expert teleoperated data. During LAM pretraining, we alternate between gradient updates on batches of unlabeled data for training the LAM and batches of labeled data for training the action decoder. The final training objective for CLAM is $\mathcal{L}_{\text{CLAM}} = \mathcal{L}_{\text{recon}} + \beta \mathcal{L}_{\text{action-decoder}}$ where $\mathcal{L}_{\text{action-decoder}} = \text{MSE}(\hat{a}_t, a_t)$ and β is a hyperparameter that balances the reconstruction and action decoder losses.

5.4.2 Latent Action Policy Training

During **Stage 2**, we use the latent IDM from our pretrained CLAM to annotate $\mathcal{D}_{\text{unlabeled-expert}}$ with latent actions. The latent FDM only provides the learning signal for training the latent IDM and is discarded at this point. We apply the latent IDM to infer the latent action z_t between each consecutive observation (o_t, o_{t+1}) , i.e., $\mathcal{D}_{\text{relabelled-expert}} = \{(o_1^i, z_1^i, o_2^i, z_2^i, \dots, z_{T-1}^i, o_T^i) \mid \forall \tau^i \in \mathcal{D}_{\text{unlabeled-expert}}\}$.

Subsequently, we train a **latent action policy**, $\pi_\theta(z_t \mid o_t)$, using imitation learning by optimizing $\mathcal{L}_\pi = \text{MSE}(\hat{z}_t, z_t)$ on batches of annotated data from $\mathcal{D}_{\text{relabelled-expert}}$. During inference time, detailed in [Algorithm 4](#), our learned policy predicts the latent actions given an observation, which the action decoder will decode into an environment action.

Leveraging pretrained IDM image features for policy training. A side-effect of learning a LAM on image-based observations is that the IDM’s image encoders can be used as pre-trained image features for learning the latent action policy (**Stage 2**). Indeed, viewed through this lens, training the LAM can be seen as a form of self-supervised representation learning. In DynaMo [65], it is shown that pretraining vision encoders using an IDM/FDM self-supervised loss improves the performance of downstream imitation learning from expert-labeled demonstrations. We will show in [Section 5.5](#) that similar positive transfer also occurs for CLAM when learning latent action policies.

5.5 Experimental Setup

Environments and Datasets. We compare CLAM to several state-of-the-art baselines using both state- and image-based observations across tasks in DMControl [284], MetaWorld [325], and on a real WidowX robot arm outlined below. Full details of the data split is provided in [Table C.13](#).

- **DMControl.** We evaluate on two DMControl locomotion tasks: HOPPER and HALFCHEETAH. We use trajectories from medium-expert split of the D4RL [91] benchmark for pre-training CLAM and subsample only suboptimal trajectories for $\mathcal{D}_{\text{labeled}}$.

Table 5.1: MetaWorld State-Based Results. We report normalized returns for DMControl tasks and average task success rates for MetaWorld tasks. **Maroon** highlights the best method in each environment, excluding **BC-Expert**, which is trained with expert labeled data. Our methods, denoted by an asterisk (*), outperform all baselines across tasks.

	HALF CHEETAH	HOPPER	ASSEMBLY	BIN PICKING	PEG INSERT	SHELF PLACE	Average
BC-AL	0.22 $\pm$ 0.05	0.35 $\pm$ 0.04	0.34 $\pm$ 0.05	0.27 $\pm$ 0.12	0.29 $\pm$ 0.07	0.00 $\pm$ 0.00	0.24
LAPD	0.12 $\pm$ 0.05	0.24 $\pm$ 0.03	0.15 $\pm$ 0.04	0.02 $\pm$ 0.03	0.17 $\pm$ 0.04	0.06 $\pm$ 0.08	0.13
LAPA	0.22 $\pm$ 0.05	0.30 $\pm$ 0.06	0.24 $\pm$ 0.07	0.15 $\pm$ 0.01	0.25 $\pm$ 0.02	0.12 $\pm$ 0.02	0.21
DynaMo	0.18 $\pm$ 0.03	0.22 $\pm$ 0.02	0.10 $\pm$ 0.03	0.06 $\pm$ 0.03	0.12 $\pm$ 0.04	0.08 $\pm$ 0.02	0.13
VPT	0.32 $\pm$ 0.04	0.41 $\pm$ 0.03	0.40 $\pm$ 0.08	0.05 $\pm$ 0.02	0.49 $\pm$ 0.06	0.02 $\pm$ 0.00	0.28
MLP-CLAM*	0.64 $\pm$ 0.05	0.64 $\pm$ 0.03	0.53 $\pm$ 0.04	0.68 $\pm$ 0.05	0.58 $\pm$ 0.04	0.72 $\pm$ 0.04	0.63
Trans.-CLAM*	0.72 $\pm$ 0.04	0.81 $\pm$ 0.05	0.91 $\pm$ 0.03	0.82 $\pm$ 0.03	0.79 $\pm$ 0.07	0.93 $\pm$ 0.02	0.83
BC-Expert	0.68 $\pm$ 0.02	0.76 $\pm$ 0.04	1.00 $\pm$ 0.00	0.94 $\pm$ 0.05	0.91 $\pm$ 0.03	0.93 $\pm$ 0.00	0.87

- **MetaWorld.** We evaluate on four difficult MetaWorld tasks: ASSEMBLY, BIN PICKING, PEG INSERT, and SHELF PLACE. We train single-task RL agents and collect the replay buffer data for pretraining. We holdout a separate random-medium dataset similar to DMControl for $\mathcal{D}_{\text{labeled}}$.
- **WidowX Robot Arm.** We evaluate on four real world manipulation tasks (see the right panel of Figure 5.3): REACH BLOCK, PUSH BUTTON, CLOSE MICROWAVE, and PUT OBJECT IN POT AND SLIDE POT which requires more than 150 timesteps at a 5 Hz control frequency. Our WidowX robot arm setup is shown in Figure 5.3 (Left). We collect a task-agnostic, play dataset of $\sim 50\text{k}$ transitions for $\mathcal{D}_{\text{unlabeled}}$ and use about 5k transitions for $\mathcal{D}_{\text{labeled}}$. We separately collect ~ 30 expert demonstrations per task without action labels for $\mathcal{D}_{\text{unlabeled-expert}}$.

Baselines. To ensure fair comparison, we reuse the same architectural components between CLAM and all baselines. These components are an IDM, an FDM, an action decoder/action head, and a transformer BC policy. Furthermore, all methods have access to same data when applicable. Some algorithms are unable to leverage certain subsets of the data, e.g. BC cannot use $\mathcal{D}_{\text{unlabeled}}$ since it requires action supervision. We compare CLAM to the following baselines:

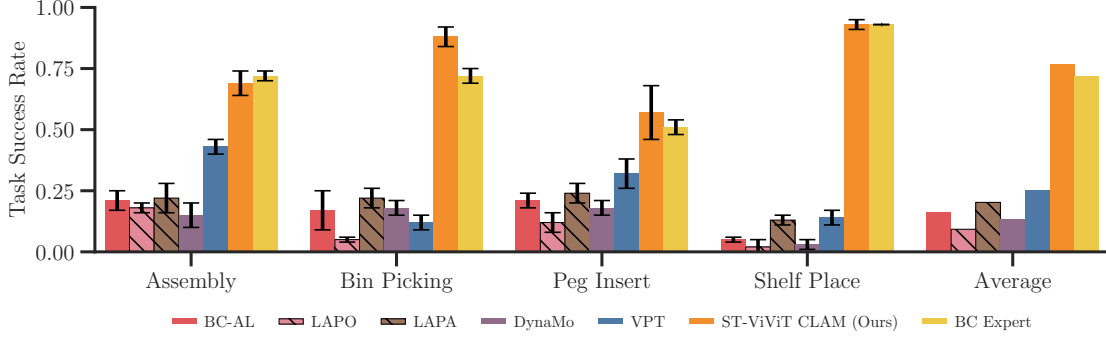


Figure 5.2: MetaWorld Image-Based Experiments. Task success rates are averaged over 50 evaluation rollouts across 3 random seeds using image-based inputs. All methods are trained with the same amount of action-labeled data (100 trajectories) for fair comparison. Because $\mathcal{D}_{\text{labeled}}$ is not necessarily expert data, baselines struggle to learn a performant downstream policy, whereas **MLP-CLAM** and **Trans.-CLAM** achieve up to a 3 $\times$ improvement over the best baseline. Baselines using a *discrete* latent action space are indicated with hashed markers. For reference, we also report results for **BC-Expert**, trained on the same number of *expert* trajectories to illustrate the ideal performance BC can achieve with ground-truth action labels.

- **BC-Action-Labeled (BC-AL)**: Behavior cloning on the small $\mathcal{D}_{\text{labeled}}$, which is not fully expert data. Since BC needs action labels, it does not use $\mathcal{D}_{\text{unlabeled}}$.
- **VPT [17]**: The IDM is trained *only* on $\mathcal{D}_{\text{labeled}}$ via supervised learning. The IDM is used to label $\mathcal{D}_{\text{unlabeled-expert}}$ with environment actions and a BC policy is then trained on the annotated data.
- **LAPO [256]**: Latent action model with discrete, vector-quantized latent actions.
- **LAPA [319]**: After LAM pretraining, the final layer of the IDM is replaced with an action head and fine-tuned end-to-end on $\mathcal{D}_{\text{labeled}}$, which is *non-expert*.
- **DynaMo [65]**: Self-supervised learning on $\mathcal{D}_{\text{unlabeled}}$ to train a vision encoder using an IDM and FDM to predict latent embeddings of future frames. A BC policy is trained on the image embeddings produced by the pretrained vision encoder using $\mathcal{D}_{\text{labeled}}$. We include this baseline to compare what a pure image-feature pretraining approach can achieve in our problem setting, which does not use the LAM for relabeling.
- **MLP-, Transformer-, ST-ViViT-CLAM (Ours)**: Continuous latent action model with different parameterizations of the latent IDM and FDM.
- **BC-Expert (BC-E)**: Privileged BC on $\mathcal{D}_{\text{unlabeled-expert}}$ with *ground-truth* action labels avail-

able, which are not available to other methods.

CLAM Model Architectures: For image observations, we model after the Space-Time (ST) Transformer [26]. We first patchify a $64 \times 64 \times 3$ image with a patch size of 16 for a total of 16 patches. Each patch is embedded through a linear layer into the hidden dimension. The encoder consists of N_E layers of Space-Time (ST) Attention blocks. Each ST block consists of spatial attention followed by temporal attention and a feedforward layer with skip connection, LayerNorm, and dropout applied between each attention. The decoder ST block also applies a cross-attention with the latent actions generated by the encoder. We add an additional token in the sequence of patch embeddings as a CLS token for the whole image. From the CLS token for each timestep, we apply a linear layer to predict the latent actions.

Latent Policy Architecture: For our policy, we use a transformer decoder similar to the one used in ACT [338]. For both external and over-the-shoulder RGB images, we use a pretrained ResNet to first extract 7×7 feature maps and flatten these features across the spatial dimension to create a sequence of d_v dimension tokens where d_v is the output dimension of ResNet. In particular, we use ResNet18 where $d_v = 512$. We feed as input to a causal transformer decoder a sequence learnable action tokens with dimension d . We use the flattened image feature map as the keys and values and apply a cross-attention between the image features and learnable tokens. We concatenate all modality tokens and add additional modality-specific embeddings and sinusoidal positional embeddings.

The input sequence to the transformer is a fixed position embedding, with dimensions $k \times 512$ where k is the chunk size and the keys and values are the combined image tokens from the stem. At each timestep, we predict a chunk of 5 actions corresponding to 1 second of execution.

5.6 Results

We aim to study the efficacy of CLAM as a general approach to learn from actionless data, evaluate its ability to train robot policies without access to labeled expert data, and analyze its design choices and limitations. We organize our experiments to answer the following:

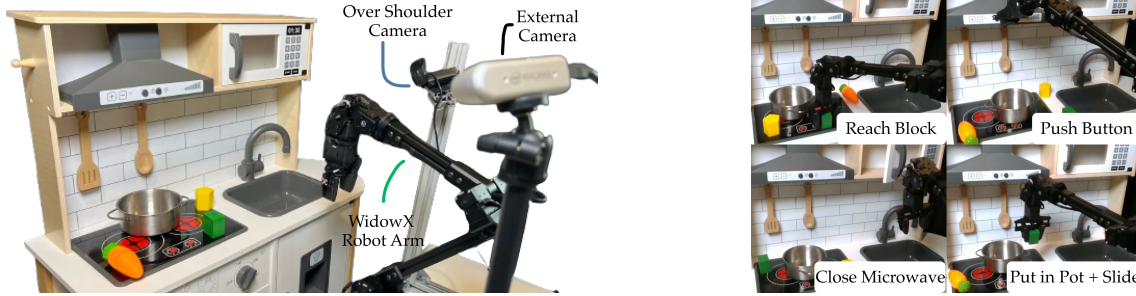


Figure 5.3: WidowX Robot Arm Setup and Evaluation Tasks. *Left:* WidowX robot arm in a toy kitchen setup [292]. We use an Intel Realsense D435 RGBD camera as a fixed external camera and a Logitech C920 webcam as an over-the-shoulder camera view. *Right:* Four tabletop manipulation tasks used to evaluate the scalability of CLAM.

- (Q1) How effective is CLAM at learning policies *without* action-labeled expert demonstrations?
- (Q2) How important is each component of CLAM: *continuous* latent actions and *jointly* training action decoder?
- (Q3) Can CLAM scale to efficiently learn capable robot policies for real-world scenarios?

FINDING 1: CLAM outperforms all baselines and nearly matches the performance of BC with expert data in both state- and image-based experiments. Table 5.1 summarizes our results for state-based inputs on DMControl and Figure 5.2 for image-based results in MetaWorld. CLAM improves upon the best baseline VPT by more than 2× average normalized return on the DMControl (*locomotion*) tasks and around 2–3× success rate on the MetaWorld (*manipulation*) tasks.

BC-AL using *action-labeled* data unsurprisingly does not perform well due to imitating sub-optimal demonstrations. In several tasks, Trans.-CLAM achieves performance close to or even better than that of BC-Expert which uses the same amount of privileged *expert* action-labeled data. In the image domain, we hypothesize that transfer from the pre-trained IDM image encoder might cause these improvements (cf. Section 5.4.2). For state-based inputs, we hypothesize that the additional difficulty introduced by not training on ground-truth actions could regularize our method and reduce overfitting.

All variants of CLAM outperform the best baseline VPT [17], highlighting the fact that latent

action models scale with $|\mathcal{D}_{\text{unlabeled}}|$ while supervised IDMs only scale with $|\mathcal{D}_{\text{labeled}}|$. Since our problem setup assumes $|\mathcal{D}_{\text{labeled}}| \ll |\mathcal{D}_{\text{unlabeled}}|$, it is likely that **VPT** learns a suboptimal IDM, underscoring the benefit of *latent* action models which can leverage vast, unstructured observation data to learn latent actions in an unsupervised manner. CLAM outperforms state-of-the-art methods in our problem setting where only *play* data is available as action-labeled data, and expert data is actionless. In other data settings, the baselines will likely be competitive, and thus choosing the right method for learning is dependent on the specific data regime. We emphasize that our data regime enables scalable learning from easy-to-collect, cheap play data [214] avoiding the need for expensive task-specific data collection.

FINDING 2: Continuous latent actions and joint action decoder greatly improve performance in real-world robotics tasks. First, we corroborate the findings by Nikulin et al. [228] that continuous latent actions joint training benefit LAM performance. Unlike **VPT**, the other baselines (**LAPQ**, **LAPA**, and **DynaMo**) make use of LAMs, as does our method. We find that we can outperform these methods, likely due to using continuous latent actions in conjunction with jointly training an action decoder. First, baselines that apply vector quantization [287] to discretize the latent actions, including **LAPQ** and **LAPA**, perform poorly on continuous control tasks. In our image-based experiments, **ST-ViViT-CLAM** achieves an over $3\times$ improvement in task success rate at 76%, compared to **LAPQ** and **LAPA**, which achieve 9% and 20%, respectively (cf. Figure 5.2). VQ discretizes the latent action space by mapping each continuous latent action to the closest vector in a learned codebook of vector embeddings. Prior works utilize VQ primarily to simplify the structure of the latent action space which is a reasonable choice for discrete action environments. We hypothesize that applying quantization to the latent space severely limits the expressivity of the latent actions for fine-grained manipulation tasks.

A potential issue with using an arbitrary continuous latent action space is grounding actions in the environment. In discrete settings, LAMs can recover an action space corresponding to a permutation of the ground-truth environment actions [42]. However, learning this mapping is

Table 5.2: CLAM Ablation Study. Both continuous latent actions and joint training are necessary to improve task success. Using continuous actions (row 2 $\rightarrow$ row 3) yields about a 1.5 $\times$ improvement in success rates. Joint training (row 3 $\rightarrow$ row 4) further boosts performance by roughly 3 $\times$.

	ASSEMBLY	BIN PICK	PEG INSERT	SHELF PLACE
Disc., w/o Joint Training	0.15 $\pm$ 0.03	0.12 $\pm$ 0.02	0.18 $\pm$ 0.04	0.19 $\pm$ 0.03
Disc., w/ Joint Training	0.14 $\pm$ 0.04	0.14 $\pm$ 0.03	0.17 $\pm$ 0.03	0.16 $\pm$ 0.04
Cont., w/o Joint Training	0.28 $\pm$ 0.04	0.18 $\pm$ 0.03	0.21 $\pm$ 0.08	0.26 $\pm$ 0.06
Cont., w/ Joint Training	0.69 $\pm$ 0.05	0.82 $\pm$ 0.04	0.57 $\pm$ 0.11	0.88 $\pm$ 0.02

more challenging for continuous actions. In Table 5.2, we present additional experiments, ablating both the choice of action space (discrete vs. continuous) and joint training on MetaWorld tasks. Using discrete latent actions, equivalent to LAPO [256], results in an average of 16% task success compared to using continuous latent actions, which achieves 23%. This indicates that even without joint training, continuous actions improve performance. Furthermore, while joint training does not help much in the discrete latent action case, we see a substantial improvement when coupled with continuous latent actions, achieving 74% average success rate (over 3 $\times$ improvement).

Real WidowX Robot Arm Experiments. We evaluate the scalability of CLAM to more realistic applications on a physical WidowX robot arm shown in Figure 5.3 (Left). Even though D_{labeled} comprises of *task-agnostic, mixed expertise play* data, we demonstrate that CLAM is still able to learn to solve the tasks while baseline methods struggle to leverage this data to train a task-specific policy. As in the simulated experiments, we find that VPT is the closest baseline, but still struggles as it can only scale with the limited amount of labeled data, while BC-AL and LAPA are unable to learn from *play* data entirely. Remarkably, CLAM learns a policy to solve new tasks, without having explicitly collected action-labeled, expert demonstrations.

This finding validates prior findings of benefits from action supervision during LAM training, as well as continuous latent action spaces in simulation settings [228] and smaller-scale experiments [316].

FINDING 3: CLAM successfully learns without ever accessing *action-labeled expert*

Table 5.3: Real Robot Results. **ST-CLAM** significantly outperforms baseline methods across all tasks. We report task success rate out of 10 trials; partial success is provided for completing a subtask.

	BLOCK	BUTTON	MICROWAVE	SLIDE POT
BC-AL	0/10	0.5/10	1/10	0/10
LAPA	2/10	3/10	3/10	0/10
VPT	2.5/10	4/10	5/10	2/10
ST-CLAM	7/10	8.5/10	8/10	4/10
BC-Expert	7.5/10	8/10	7.5/10	2/10

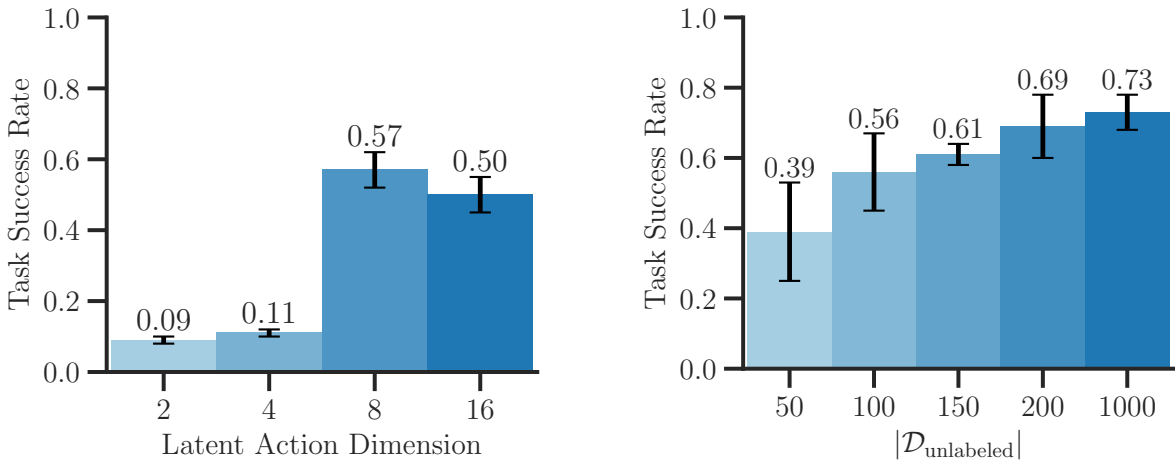


Figure 5.4: (Left) Increasing the latent action dimension improves the model expressivity for policy learning. Up until a latent dimension of 4, the learned latent action space fails to be useful for imitation learning. However, a latent dimension of 8 has sufficient capacity, achieving 57% success rate on the Assembly task. **(Right)** CLAM scales with the amount of unlabeled video data. The performance of the downstream policy improves as we annotate more trajectories using the pretrained CLAM.

data in both simulation and real robot experiments. We perform our main-line experiments (Figure 5.2) with action-labeled data that is not fully expert. This type of data is much cheaper to collect than training human workers to teleoperate robot hardware with potentially many degrees of freedom [207]. **LAPA** circumvents the issues of learning a separate action-grounding model by directly fine-tuning the pretrained LAM with an uninitialized action prediction head on $\mathcal{D}_{\text{labeled}}$. However, if only partially-optimal or play data is available, the fine-tuned BC model struggles to learn a good policy, since it does not label the unlabeled data. In this regard, **LAPA** is similar to **BC-AL** with a better initialization as a result of the LAM pretraining, (20% vs. 16%

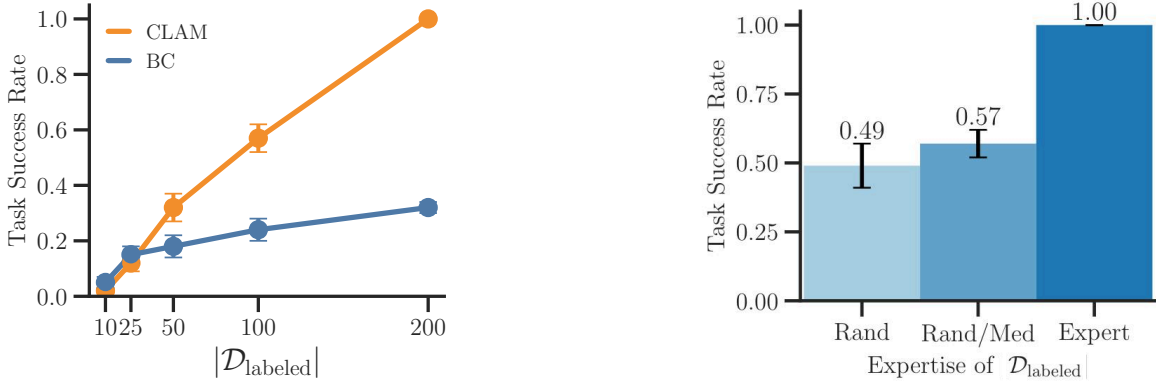


Figure 5.5: (Left) We vary the number of labeled trajectories for training the action decoder. While BC performance struggles to learn from *non-expert data*, our method improves with more data. **(Right)** We also evaluate the robustness of CLAM to varying expertise of data. We learn a better policy than BC with the same amount of labeled random trajectories. Unsurprisingly, with expert data, our method recovers an optimal policy.

success rate). Despite the action-labeled data not being fully optimal, CLAM is still able to achieve high success rates.

To further investigate this capability of our method, we experiment with various data compositions composed of varying expertise levels. We filter our offline dataset for 100 trajectories below a predefined threshold return value and manually verify that the policy does not achieve the downstream task and acts randomly. We show in Figure 5.5 (Right) that even with random policy data we can achieve similar performance as on the random/medium data regime. When expert data is available, we can recover a policy that always solves the task.

5.6.1 CLAM Design Choices

We conduct a comprehensive analysis of various design choices such as the latent action dimension, the amount of unlabeled data ($|\mathcal{D}_{\text{unlabeled-expert}}|$) for latent policy training, and the amount of labeled data ($|\mathcal{D}_{\text{labeled}}|$) for action decoder training.

Latent action dimension directly affects the model’s expressivity. In Figure 5.4 (Left), we vary the latent action dimension $|z| \in \{2, 4, 8, 16\}$ for the MetaWorld Assembly task. We find that setting $|z|$ to the true action dimension (4 in the case of MetaWorld) is insufficient,

likely because our LAM is not guaranteed to learn the same compact action representation the environment uses. We find that having a slight overparameterization for the latent action space makes learning easier. Between $|z|$ of 4 and 8, there is a significant improvement. However, further increasing $|z|$ to 16 does not yield any additional gains. This suggests that a latent dimension of 8 is sufficient to capture the representational capacity needed for policy learning. Higher action dimensions are likely more difficult to learn by the action decoder, potentially requiring more labeled data.

Latent action policy scales with $|D_{\text{unlabeled-expert}}|$. In Figure 5.4 (Right) we demonstrate that CLAM’s performance on the Assembly task improves as we increase $|D_{\text{unlabeled}}|$. This result suggests that we can improve robot policies without the need for expensive, manually collected expert teleoperated demonstrations. We note that the returns start diminishing after a certain number of trajectories, likely because the action decoder’s data remains unchanged.

Increasing $|D_{\text{labeled}}|$ improves the action decoder accuracy and downstream performance. In Figure 5.5 (Left) we analyze the effect of varying $|D_{\text{labeled}}|$ for training the action decoder. Unsurprisingly, as we increase $|D_{\text{labeled}}|$, the learned action decoder becomes more accurate and better generalizes to new unseen states. With only a handful of trajectories, it is difficult to ground the latent actions to the environment explaining the poor performance. On the contrary, BC trained using the same amount of labeled *non-expert* data quickly plateaus in performance and fails to scale with more data. In our problem setting, with 50 non-expert labeled trajectories, CLAM outperforms baselines, including BC, trained on more than 100. With enough labeled trajectories, our latent action space is expressive enough to achieve perfect performance on the task.

5.7 Conclusion

We proposed CLAM, a scalable solution for learning continuous control tasks from unlabeled video data. We demonstrate with real robot experiments that CLAM, using continuous latent actions and joint training, can learn policies for unseen tasks without requiring any expert tele-

operated demonstrations. We hope CLAM further advances the scalable training of robot policies from action-less data, alleviating the need for expensive, manual data collection.

5.8 Limitations

Embodiment Gap. The human-robot embodiment gap is directly addressable with video conversion methods such as Phantom [177] and Masquerade [176]. We leave performing studies that use these or similar approaches in the actual pipeline for future work.

Alternatively, other approaches to addressing the human-robot embodiment gap. Prior works such as CrossFormer [73] train fine-tuned action heads for different embodiments, but are limited to a fixed set of embodiments seen during training. For future work, we aim to apply the action-less learning capabilities of CLAM to the cross-embodiment problem, to allow learning for various robot embodiments and the human embodiment in a scalable way.

CHAPTER 6

Chapter 6

Inference-Time Plan Guidance with Interleaved Reasoning

6.1 Introduction

Reasoning models [103, 136, 313] typically take a “think-then-answer” approach, in which a monolithic, and often quite long, block of reasoning is generated before producing any user-facing output. This delay not only creates a poor user experience, but can result in substantial wasted time if it misunderstands the user’s intent or begins its reasoning from a flawed assumption. Because it offers no opportunity for user intervention, the “think-then-answer” paradigm is especially problematic in time-sensitive applications like voice assistants.

Human speakers avoid this problem by employing a range of strategies for *collaborative grounding*, which help to ensure mutual understanding throughout dialogue [25, 58, 259]. Motivated by this literature, we introduce *interleaved reasoning*, in which the model alternates between unobserved “thinking” and surfacing intermediate responses to the user. We define an intermediate response as a self-contained, usable piece of information that hints at the model’s understanding of the user’s intention and its plan for satisfying it. When this understanding or plan is incorrect, it should be possible for the user to act immediately without waiting for a future revision. For example, in response to a trip planning prompt, the model might first pro-

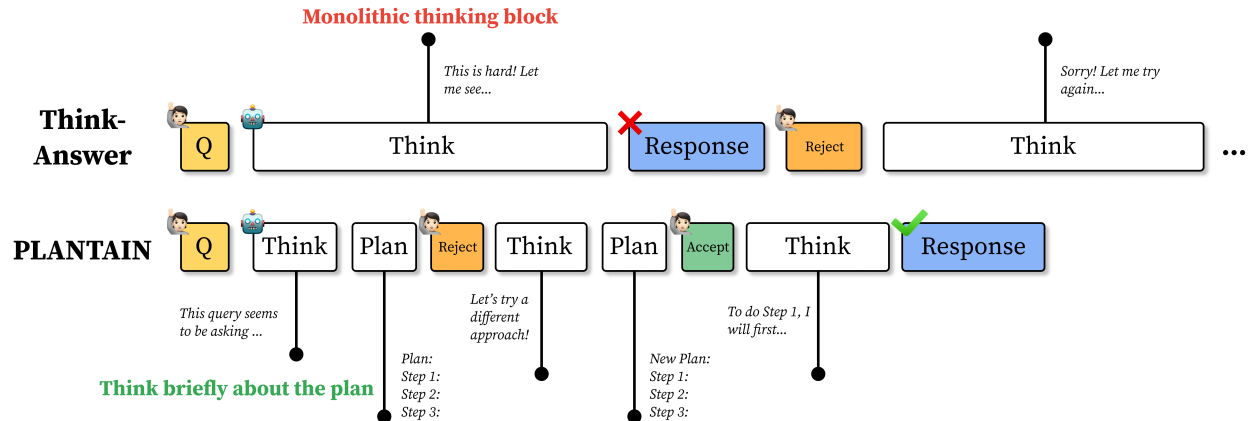


Figure 6.1: PLANTAIN Rewind-and-Repeat After post-training an interleaved reasoning model, we apply an iterative rejection-sampling strategy for inference-time plan guidance. PLANTAIN first produces a *plan* as its initial intermediate response. This plan is evaluated by a human judge or an LLM autorater. If accepted, the model proceeds to generate the final answer. If rejected, the process is *rewound* such that the rejected plan is appended to a history of failures, and the model is re-prompted to produce a new, distinct plan. Crucially, the response generated by PLANTAIN is typically of higher quality than the response produced by a standard Think–Answer model. Additionally, subsequent thinking blocks after a rejection plan are short because the model only needs to adjust the plan rather than regenerate full reasoning. This leads to early pruning of suboptimal reasoning paths and closer alignment with true user intent.

vide a high-level outline of a possible itinerary. Subsequent responses could then provide more granular, day-to-day activities, after making additional web searches and tool calls.

We propose PLANTAIN (**P**lan-**A**nswer **I**nterleaved **R**easoning), a post-training framework that enables existing models to perform interleaved reasoning. PLANTAIN follows a three-stage recipe: (1) generating synthetic dataset of interleaved reasoning traces using prompting (2) supervised fine-tuning to distill the desired format into the base model, and (3) reinforcement learning (RL) post-training with verifiable rewards for improving downstream task performance. In this chapter, we focus on a “plan-first” specialization of interleaved reasoning, shown in Figure 6.1, where the model’s initial intermediate response is an explicit, *step-by-step* plan that verbalizes its intended solution path.

While prior work has shown that grounding model generation in a plan improves performance [318], our interleaved approach uniquely allows for early intervention to keep the solution on track. To this end, we design two inference-time strategies, **Best-of-N-Plans** and **Rewind-and-Repeat**, that use an LLM-as-a-judge to simulate user feedback on the initial plan(s). In the

Best-of-N-Plans approach, the model generates multiple plans and the user selects the best one for the subsequent generation. In Rewind-and-Repeat [153], the model proposes one plan at a time, which the user can accept or reject; if a plan is rejected, the process is “rewound”, and the model is prompted to generate a new, different plan. We evaluate these methods on a diverse set of benchmarks, including math reasoning, coding, text-to-SQL structured generation, and long-context question answering. Our proposed strategies improve alignment and robustness without increasing total token cost, since only short plan prefixes are resampled rather than full generations. This allows the model (or user) to prune flawed reasoning paths early, yielding higher pass@1 and lower latency compared to standard Best-of-N over complete responses [261].

Our core contributions are as follows:

1. We introduce PLANTAIN, a framework for “plan-first” interleaved reasoning, along with two inference-time strategies that allow for early user intervention and feedback.
2. We demonstrate that PLANTAIN reduces the time-to-first-token by **60%**, which we use as a measure of perceived latency, without compromising task performance.
3. We demonstrate that PLANTAIN—when trained exclusively on coding data—generalizes to diverse reasoning tasks, achieving an average **+6%** improvement in pass@1 across all benchmarks.

6.2 Interleaved Reasoning

PLANTAIN is a post-training framework to elicit interleaved reasoning behavior in reasoning models. Unlike prior approaches that impose length restrictions or introduce token budgets [4, 110, 262], PLANTAIN encourages the model to produce an explicit *plan* as the first intermediate response. This behavior is controlled through the system instruction (SI), which is modified to prompt the model to “plan first,” while the SFT and RL objectives reinforce the resulting interleaved reasoning style without explicitly constraining when to plan. Our framework is a simple three-step recipe: (1) generation of synthetic interleaved reasoning traces, (2) supervised fine-tuning to distill the interleaved format, and (3) RL post-training using verifiable rewards. Having

obtained models capable of interleaved reasoning, we then introduce two inference-time strategies that leverage this output structure to improve the final response generation. The complete pseudocode for our training and inference procedures is provided in [Section D.1](#).

Interleaved CoT. A standard language model is typically trained to produce a final answer A conditioned on a prompt P , often preceded by a monolithic block of reasoning, T , resulting in a sequence of the form, $P \rightarrow T \rightarrow A$. Rather than outputting the final answer after a lengthy thought block, interleaved CoT alternates between shorter thoughts and outputting an intermediate response to the user. Formally, an interleaved trace $\tau = (t_1, a_1, t_2, a_2, \dots, t_n, a_n)$ where t_i represents internal thoughts and a_i is an intermediate response surfaced to the user.

Generating Synthetic Interleaved Traces. Off-the-shelf reasoning models exhibit a strong bias towards the “think-answer” paradigm, an artifact of their training that results in lengthy reasoning sequences even for simple prompts [51]. Following [144], we iteratively prompt a larger model to produce variable length CoT traces. We take a subset of prompts from BIG-CODEBENCH [347] and create a natural response decomposition, by modifying the prompt ([Section D.3.1](#)) to ask for a solution outline and unit tests in addition to the code implementation. Concretely, each interleaved CoT trace follows the structure *thought* $\rightarrow$ *solution plan* $\rightarrow$ *thought* $\rightarrow$ *code* $\rightarrow$ *thought* $\rightarrow$ *unit tests*, encouraging the model to emit useful early outputs for the user. We provide an example of the generated interleaved traces in [Appendix D.3.1](#). To improve generalization, we further construct (i) *concatenated-prompt traces*, where multiple independent prompts are concatenated into a composite input, and the model is asked to solve each sequentially within a single reasoning trace, and (ii) *multi-solution traces*, where the model is prompted to generate several distinct candidate solutions for the same problem. For example, a concatenated-prompt trace may include three standalone coding tasks, such as reversing a string, counting unique elements, and checking for palindromes, encouraging the model to transition between problems within one reasoning context. A multi-solution trace could prompt the model to produce multiple implementations for a single task (e.g., recursive and iterative factorial). Together, these examples form our synthetic interleaved reasoning dataset, denoted as $\mathcal{D}_{\text{interleave}} = \{(P_j, \tau_j)\}_{j=1}^N$. This

Algorithm 5 Post-training to elicit interleaved reasoning

Require: Base model π_{θ_0} ; prompts $\mathcal{X}$; synthetic “plan-first” prompt template $\text{SI}_{\text{plan-first}}$; synthetic generator Π^* ; weights α_{fmt} , α_{succ} ; number of RL updates K .

Ensure: Interleaved model π_{θ}

```
1: // Construct synthetic interleaved trace dataset.
2:  $\mathcal{D}_{\text{interleave}} \leftarrow$ 
    $\{(x', \tau) \mid x' = \text{SI}_{\text{plan-first}}(x), \tau \sim \Pi^*(\cdot \mid x'), x \in \mathcal{X}\}$ 
3: // SFT on the synthetic interleaved trace dataset.
4:  $\text{SUPERVISED FINE TUNE}(\pi_{\theta_0} \rightarrow \pi_{\theta}; \mathcal{D}_{\text{interleave}})$ 
5: // RL post-training to obtain PLANTAIN model.
6: for  $k = 1$  to  $K$  do
7:   Sample  $(x', \cdot) \sim \mathcal{D}_{\text{interleave}}$ ; rollout  $y \sim \pi_{\theta}(\cdot \mid x')$ 
8:    $r_{\text{fmt}} \leftarrow \text{FORMATOK}(y)$ 
9:    $r_{\text{succ}} \leftarrow \text{FORMATOK}(y) \cdot \text{TASKSUCCESS}(x', y)$ 
10:   $r \leftarrow \alpha_{\text{fmt}} r_{\text{fmt}} + \alpha_{\text{succ}} r_{\text{succ}}$ 
11:   $\text{PERFORM RL UPDATE}(\theta; x', y, r)$ 
12: return  $\pi_{\theta}$ 
```

construction diversifies the amount of multi-step interleaving in our training data and prevents the model from overfitting to a fixed output template.

We fine-tune the base model on $\mathcal{D}_{\text{interleave}}$ by minimizing the negative log-likelihood. This process effectively “distills” the desired interactive behavior from our synthetic data into the model, shifting its default response style from monolithic to interleaved. We then post-train the supervised fine-tuned model using Proximal Policy Optimization (PPO) [257]. PPO utilizes a value network to approximate the state-value function and Generalized Advantage Estimation to compute the advantage function. (Xie et al. [305] found PPO to be more stable during training compared to GRPO [103] because of its extra critic model.) The policy model π_{θ} generates rollouts that maximize an expected reward $\mathbb{E}[r(x, y)]$.

Reward function. For our modified coding prompts, we define a composite reward comprised of four rule-based components: a **format** reward that checks whether the response correctly interleaves multiple intermediate answers, a **correctness** reward based on the pass rate of the generated code against golden unit tests, a **helpfulness** reward produced by an LLM-as-a-judge autorater evaluating the quality of the outline, and a **unit-test** reward indicating whether a valid unit-test block was produced. The overall reward is computed as: $r_{\text{interleave}}(x, y) =$

$\mathbb{I}_{\text{fmt}}(y)(1 + r_{\text{corr}}(y) + r_{\text{help}}(y) + \mathbb{I}_{\text{unit}}(y))$, where

$$r_{\text{format}}(y) = \mathbf{1}\{\text{response contains all of the required sections in the correct order}\},$$

$$r_{\text{corr}}(y) = \frac{\# \text{ tests passed}}{\# \text{ total tests}},$$

$$r_{\text{helpful}}(y) = \text{LLM-Judge}(x, y) \in [0, 1],$$

$$r_{\text{unit}}(y) = \mathbf{1}\{\text{valid unit-test block detected}\}.$$

By including the indicator $r_{\text{format}}(y)$ as a multiplicative term, we ensure that downstream rewards are applied only when the interleaved format is satisfied. In preliminary experiments, we find that naively modifying the System Instruction (SI) does not induce the desired interleaved behavior. Models prompted in this way often collapse the plan generation into the reasoning trace and revert to the standard monolithic *think-then-answer* format. This structure is essential for eliciting consistent plan-first reasoning and enabling our subsequent inference-time strategies.

6.3 Inference-Time Scaling

Given a trained interleaved reasoning model, we can apply inference-time strategies to better align model outputs with user intent, pruning erroneous reasoning paths early in the generation process. These strategies operate at inference without modifying the underlying model weights, leveraging the model’s ability to produce an explicit *plan* as the first intermediate response.

While we envision these approaches having utility in interactive settings with human users, in this chapter we simulate the human with an autorater that provides feedback on candidate plans (full prompt shown in [Section D.2.2](#)). The judge receives as input the user’s original prompt p and a proposed plan a_i ; because it has access to the user prompt and plan but not the model’s internal reasoning trace, it evaluates plans solely from the perspective of an external user. Given N candidate plans $\{p_i\}_{i=1}^N$, the judge is prompted to either (i) select the index of the best plan when $N > 1$, or (ii) output a binary decision $\{\text{accept}, \text{reject}\}$ when evaluating a single plan ($N = 1$).

Given that our model can interleave between internal thinking and user-facing responses, we

Algorithm 6 Best-of-N-Plans

Require: Interleaved model π_θ ; prompt p ; temperature τ ; number of plans N ; AutoRater R for scoring plans.

Ensure: Final response y .

- 1: // Convert p to a “plan-first” prompt.
 - 2: $p' \leftarrow \text{SI}_{\text{plan-first}}(p)$
 - 3: // Sample and score N plans.
 - 4: $\{(a_i, s_i)\}_{i=1}^N \leftarrow \{(a_i, R(p, a_i)) \mid a_i \sim \pi_\theta(a \mid p'; \tau)\}$
 - 5: // Select the best auto-rated plan.
 - 6: $a^* \leftarrow \arg \max_{a_i} s_i$
 - 7: // Complete rollout according to the plan, and answer.
 - 8: $y \sim \pi_\theta(\cdot \mid p', a^*)$
 - 9: **return** y
-

use the first intermediate output as a *plan*. We introduce two inference-time control methods based on this plan structure: (1) *Best-of-N-Plans*, which samples multiple candidate plans and uses the LLM autorater to select the plan that best addresses the user prompt, and (2) *Rewind-and-repeat*, which generates a new plan whenever the current one is rejected, continuing until an acceptable plan is found or a retry budget is exhausted. Together, these strategies enable adaptive, user-aligned reasoning at inference time without retraining. Although the model is initially trained on coding tasks, these inference-time mechanisms are domain-agnostic and extend naturally to other reasoning settings such as mathematics, text understanding, or planning. Pseudocode for the inference-time strategies is provided in [Algorithm 6](#) and [7](#).

Best-of-N Plans. In this setting, the model generates a diverse set of N candidate plans, $\{a_1, a_2, \dots, a_N\}$, for a given user prompt p . To encourage diversity, we sample with a high temperature ($\tau > 1$), which increases the likelihood of exploring less frequent but potentially insightful reasoning paths. The generated plans are then evaluated by the LLM *autorater*, which serves as a proxy for human feedback. The autorater receives the user’s prompt p along with the set of candidate plans, but not the model’s internal reasoning process, introducing *information asymmetry* between the model and autorater. This setup mirrors real user evaluation, where one can judge whether a plan is well-structured and relevant to the prompt without access to the model’s internal thoughts. The autorater selects the plan that best addresses the prompt and is most likely to lead to a correct or complete final response, denoted as a^* . The model then continues generation

Algorithm 7 Rewind-and-Repeat (R&R)

Require: Interleaved model π_θ ; prompt p ; number of retries T ; AutoRater R for accepting or rejecting plans.

Ensure: Final response y .

```
1: // Initialize the prompt and rewind history.
2:  $p' \leftarrow \text{SI}_{\text{plan-first}}(p)$ ;  $\mathcal{H} \leftarrow \emptyset$ 
3: for  $t = 1$  to  $T$  do
4:   // Propose and rate a plan.
5:    $a_t \sim \pi_\theta(a \mid \text{Augment}(p', \mathcal{H}))$ ;  $d_t \leftarrow R(p, a_t)$ 
6:   if  $d_t = \text{accept}$  then
7:     // Accept plan, complete rollout, and answer.
8:     return  $y \sim \pi_\theta(\cdot \mid p', a_t)$ 
9:   else
10:    // Store the rejected plan in the rewind history  $\mathcal{H}$ .
11:     $\mathcal{H} \leftarrow \mathcal{H} \cup \{a_t\}$ 
12: // Fallback to a past plan from  $\mathcal{H}$  if all attempts are rejected.
13:  $a_r \sim \text{Uniform}(\mathcal{H})$ ; return  $y \sim \pi_\theta(\cdot \mid p', a_r)$ 
```

conditioned on a^* , producing the subsequent reasoning steps and final answer.

Rewind-and-repeat. Instead of generating multiple plans at once, this approach performs plan generation in an iterative feedback loop. The model first proposes an initial plan a_1 conditioned on the user prompt p . The LLM autorater then evaluates the plan and returns a binary verdict $d_1 \in \{\text{accept}, \text{reject}\}$. If accepted, the model proceeds to complete the response using a_1 . If rejected, the plan is added to the failure set $\mathcal{H} = \mathcal{H} \cup \{a_1\}$, and the model is re-prompted with the original query and rejection history. This conditioning encourages the model to generate a new plan that avoids previous failure modes. The process repeats for up to T iterations or until a plan is accepted. If all attempts are rejected, the model samples a fallback plan from $\mathcal{H}$ and continues generation from that plan.

6.4 Experimental Setup

Our experiments address the following questions:

- (Q1) Does interleaved reasoning reduce inference-time *latency* without compromising the final task performance compared to the standard think-answer approach?
- (Q2) How effective are our inference-time strategies at improving initial plan quality and pruning

suboptimal reasoning paths before execution?

Models and Baselines. We use the QWEN3 reasoning model family [313] as our base architecture, and evaluate both the 4B and 8B parameter variants. We compare our post-trained interleaved reasoning models and inference-time strategies against several baselines and ablations. The first set of baselines is applied directly to the base model without any additional fine-tuning.

- **No Thinking:** The base model is prompted to generate the answer directly, without explicit CoT reasoning.
- **Think-Answer (TA):** The base model is prompted to perform explicit reasoning steps in a monolithic block before generating the final answer.
- **Interleaved Reasoning via RL [305]:** An RL-only post-training approach that induces interleaved reasoning from a prompted base model, without a supervised fine-tuning stage on interleaved traces.
- **Rewind-and-Repeat (R&R) on Answer:** Rewind-and-repeat applied to the *final answer*. An LLM autorater judges the final response and then triggers a full restart from the original prompt if the answer is rejected. This inference strategy is equivalent to our proposed *R&R*, but applied at the final response level.

We compare the baselines to our inference-time strategies applied on the RL-trained interleaved reasoning model.

- **Plan-Answer:** Direct inference of the interleaved thinking model. First generate a plan, and then the final answer.
- **Best-of-N Plan:** Use the RL fine-tuned model to generate N plans in parallel, and select the best one using an LLM autorater. The final response is then conditioned on the selected Best-of-N plan.
- **Rewind-and-Repeat (R&R) Plan:** Iterative rejection-sampling approach that generates a single plan at a time, with an LLM autorater to decide whether to accept it and continue, or reject and "rewind" to generate a new plan.

Interleaved and Evaluation Datasets. We train on a combination of coding and mathematical reasoning datasets (BigCodeBench, MBPP, and MATH500). To assess cross-domain generalization we evaluate across two additional domains: text-to-SQL and long-context question answering. BigCodeBench [347] contains 1.1K function-level Python coding tasks requiring multi-library reasoning and compositional code synthesis. MBPP [11] consists of 974 entry-level Python problems designed for evaluating basic programming competence. MATH500 [191] includes 500 symbolic reasoning problems spanning algebra, geometry, and probability. BirdSQL [181] and QuALITY [235] are used for out-of-domain evaluation: the former tests text-to-SQL translation grounded in real relational schemas, while the latter measures long-context reading comprehension with passages averaging 5K tokens. See Section D.2.3 for further dataset and split details. We generate interleaved traces on 50 BigCodeBench prompts modified to request a solution outline and unit tests in addition to the code solution. Iterative CoT prompting on QWEN3-32B is used to synthesize these traces. We additionally collect 50 traces by concatenating pairs of MBPP prompts and 25 traces that request multiple solutions for a single coding problem, resulting in 125 total interleaved reasoning traces used for SFT.

Evaluation Metrics. We evaluate model performance using four primary metrics: **task success rate (pass@1)**, **unit test pass rate**, **time-to-first-response (TTFR)**, and **tokens-per-problem (T/P)**. Pass@1 measures the percentage of prompts for which the model produces a fully correct solution on the first attempt and is reported across all benchmarks. For coding tasks, we additionally report the unit test pass rate, the proportion of ground-truth unit tests passed by the generated code, normalized by the total number of tests. TTFR captures the number of *thought tokens* generated before the first user-visible response (i.e., the initial plan), providing a measure of response latency. While token count does not map linearly to wall-clock time, it provides a reliable model-agnostic proxy for response latency, since inference speed is approximately proportional to the number of generated tokens under a fixed decoding setup. The T/P ratio quantifies overall token efficiency as the average number of tokens generated per problem, including both internal reasoning and user-facing outputs. For interleaved traces, we further

assess intermediate response quality using an LLM-as-a-judge (QWEN2.5-7B-INSTRUCT), which labels each segment as *helpful* or *not helpful*.

For our inference-time strategies, we compute TTFR based on the first generated plan, even if it is later rejected by the autorater. When reporting T/P, we include all thought tokens from any additional plan generations. In the *Best-of-N* setting, plans are generated in parallel, so latency is bounded by the slowest plan. In contrast, the *Rewind & Repeat (R&R)* strategy generates and evaluates plans sequentially, though the short length of each thought block results in negligible overhead relative to the full rollout.

6.5 Results

Interleaved reasoning generalizes beyond coding, reducing latency and improving task accuracy across domains. Table 6.1 compares Think-Answer (TA) and Plan-Answer decoding for QWEN3-4B and QWEN3-8B across MATH500, MBPP, Text-to-SQL, and QuaLITY. Models trained to interleave planning with reasoning on coding data transfer effectively to unseen domains, substantially lowering time-to-first-response (TTFR) while maintaining or improving final task performance. For example, on MATH500, TTFR decreases from 2044 $\rightarrow$ 628 tokens for QWEN3-4B (84.2 $\rightarrow$ 84.4 P@1) and from 2106 $\rightarrow$ 625 tokens for QWEN3-8B (88.2 $\rightarrow$ 85.2 P@1), with similar trends on MBPP and QuaLITY where TTFR is reduced by over 60%. The “No Thinking” baseline yields TTFR = 0 because the model directly outputs an answer without generating any intermediate reasoning tokens; this serves as a lower bound on perceived latency but typically produces less reliable outputs. Despite being post-trained only on coding tasks, the model exhibits consistent improvements on math, text-to-SQL, and reading comprehension, indicating that explicit planning structures learned in one domain promote more efficient and accurate reasoning in others. This demonstrates that interleaved reasoning not only reduces pre-answer token overhead but also enhances generalization and response quality across diverse reasoning tasks.

Table 6.1: Interleaved reasoning generalizes beyond coding tasks, reducing latency and preserving accuracy. Plan-first decoding (*Plan-Answer*) substantially lowers time-to-first-response (TTFR) while maintaining or improving pass@1 accuracy compared to the standard *Think-Answer* baseline with a token budget of 4096. Despite being post-trained only on coding data, the learned interleaved behavior transfers effectively to math, text-to-SQL, and reading comprehension tasks.

Method	MATH500		MBPP		Text-to-SQL		QuaLITY		Average	
	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓
QWEN3-4B										
No Thinking	81.6	0	48.2	0	18.6	0	61.0	0	52.4	0
Think-Answer	82.8	1492	50.8	1298	25.4	1542	68.4	1345	56.9	1419
Plan-Answer	84.4	628	52.6	523	26.0	484	72.2	428	58.8	516
QWEN3-8B										
No Thinking	82.4	0	51.4	0	21.0	0	64.2	0	54.8	0
Think-Answer	83.0	1587	54.5	1417	28.0	1684	70.6	1463	59.0	1538
Plan-Answer	85.2	625	56.8	554	28.0	457	76.4	443	61.6	520

Table 6.2: QWEN3-8B at an 8192-token decoding budget. Pass@1 accuracy (%) on MATH500, MBPP, Text-to-SQL, and QuaLITY. We compare Think-Answer (TA) and an RL-only interleaved baseline [305] against our supervised-then-RL Plan-Answer model with inference-time strategies Best-of-N (5) and R&R (Plan). Best per column in **bold**.

Method	MATH500	MBPP	Text-to-SQL	QuaLITY	Avg.
Think-Answer (8192)	88.2	56.4	31.0	78.4	63.5
Interleaved RL [305]	87.8	57.1	31.8	77.6	63.6
Plan-Answer (ours)	85.2	56.8	28.0	76.4	61.6
Best-of-N (5, 8192) (ours)	89.4	58.7	32.5	78.8	64.9
R&R (Plan, 8192) (ours)	89.6	59.1	34.3	78.4	65.4

We also include a direct comparison to Interleaved Reasoning via RL [305], which operates in a related setting and targets improved reasoning efficiency through interleaving. To ensure an apples-to-apples comparison, we re-ran this baseline under the same backbone, task suite, and decoding budgets as in this chapter, initializing all methods from the same QWEN3-8B checkpoint. Unlike PLANTAIN, which uses supervised fine-tuning on synthetic interleaved traces before RL, this baseline relies on reinforcement learning alone to elicit interleaved behavior from a prompted base model. Table 6.2 reports pass@1 at an 8192-token budget alongside our Plan-Answer model and inference-time strategies.

Early plan-level feedback guides reasoning toward correct solution paths, yielding up to +2–3% higher task accuracy with 7× lower TTFR. Table 6.3 compares inference-time control strategies on the RL-trained interleaved model. Unlike answer-level feedback that arrives only after full reasoning, plan-level evaluation intervenes *before* subsequent reasoning begins, allowing the model to revise faulty plans and steer downstream thoughts toward more accurate outcomes. Both Best-of- N and Rewind-and-Repeat (R&R) at the plan stage outperform the base Plan-Answer decoding, showing that lightweight, inference-time guidance alone improves reasoning quality without retraining. For QWEN3-4B, R&R (Plan) boosts MATH500 accuracy from 84.4 $\rightarrow$ 86.8 while maintaining $\sim$ 578 TTFR tokens, over 7× faster than R&R (Answer). Similarly, QWEN3-8B achieves +2.2 P@1 on MATH500 and +0.6 on MBPP with TTFR $\approx$ 600. These results demonstrate that plan interleaving provides earlier, denser supervision that improves both efficiency and final task accuracy.

Table 6.3: Inference-time strategies: Plan-level feedback provides earlier and more informative supervision than answer-level feedback, yielding higher accuracy and lower TTFR. Both Best-of- N and R&R (Plan) improve over Plan-Answer without additional training.

Method	MATH500		MBPP		Text-to-SQL		QualITY		Average	
	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓	P@1↑	TTFR↓
QWEN3-4B										
Plan-Answer	84.4	628	52.6	523	26.0	484	72.2	428	58.8	516
R&R (Answer)	84.4	4079	52.4	3321	31.2	2822	74.0	2453	60.5	3169
Best-of- N (5)	86.2	523	54.6	584	28.4	568	75.2	547	61.1	556
R&R (Plan)	86.8	578	53.8	519	31.6	552	75.0	534	61.8	546
QWEN3-8B										
Plan-Answer	85.2	625	56.8	554	28.0	457	76.4	443	61.6	520
R&R (Answer)	86.0	4279	56.2	3541	34.0	2957	78.2	2614	63.6	3348
Best-of- N (5)	85.8	628	58.2	641	31.0	602	78.2	582	63.3	613
R&R (Plan)	89.4	611	57.4	543	34.0	568	78.2	562	64.8	571

Think-Answer models rely on longer reasoning chains to improve, indicating inefficient token usage, whereas interleaved models attain comparable accuracy with fewer tokens. Table 6.4 varies the available context window (4096 $\rightarrow$ 8192 tokens). While Think-

Table 6.4: Token-budget ablation: Interleaved inference achieves near-saturated accuracy at 4K tokens, highlighting efficient use of available context. While Think-Answer continues to improve with longer reasoning—reflecting redundant token usage—Plan-Answer benefits modestly from larger budgets but remains below Rewind-and-Repeat, indicating stable yet non-adaptive scaling. Rewind-and-Repeat achieves the strongest token efficiency and overall accuracy.

Method (4096 → 8192)	MATH500	MBPP	Text-to-SQL	QualITY	Average
QWEN3-4B					
Think-Answer (TA)	82.8 → 84.2	50.8 → 52.0	25.4 → 28.2	68.4 → 74.6	56.9 → 59.8
Plan-Answer	84.4 → 86.0	52.6 → 53.5	26.0 → 29.5	72.2 → 74.8	58.8 → 60.9
R&R (Plan)	86.8 → 87.2	53.8 → 54.3	31.6 → 31.7	75.0 → 75.4	61.8 → 62.2
QWEN3-8B					
Think-Answer (TA)	83.0 → 88.2	54.5 → 56.4	28.0 → 31.0	70.6 → 78.4	59.0 → 63.5
Plan-Answer	85.2 → 88.8	56.8 → 58.5	28.0 → 32.5	76.4 → 78.2	61.6 → 64.5
R&R (Plan)	89.4 → 89.6	57.4 → 59.1	34.0 → 34.3	78.2 → 78.4	64.8 → 65.4

Answer (TA) models show modest gains with larger budgets (e.g., MATH500 82.8 → 84.2), interleaved models nearly saturate at 4096 tokens (86.8 → 87.2 for QWEN3-4B; 89.4 → 89.6 for QWEN3-8B). This demonstrates that 4K tokens are sufficient for plan-conditioned inference, even without extended “thinking” budgets.

In contrast, the TA baseline exhibits poor token efficiency. Its improvements come primarily from longer reasoning sequences, much of which is spent on redundant or self-corrective thought. Empirically, many TA generations fail to terminate naturally and in these cases, the model continues reasoning until truncated, after which a final answer must be explicitly prompted. Interleaved models, by comparison, allocate tokens adaptively and terminate after a few reasoning–answer alternations, effectively regularizing total reasoning length.

In the *Rewind-and-Repeat* setting, this difference becomes particularly salient. Because interleaved models surface an explicit plan before full reasoning, each rewind operates over a concise, interpretable intermediate representation rather than an entire free-form completion. This allows the LLM-as-a-judge to prune low-quality trajectories early — avoiding wasted computation and ensuring that subsequent reasoning unfolds around a verified plan, whereas TA models can only rewind after full generations, offering no opportunity for early correction. On MATH500, for ex-

ample, the TA model frequently degenerates into repetitive or self-contradictory reasoning and fails to recover once diverged, while plan-based interleaving grounds the process through explicit subgoals that constrain and stabilize subsequent thoughts. Overall, these results show that interleaved inference regularizes reasoning depth, avoids unnecessary computation, and achieves higher token efficiency under fixed or limited context budgets. See [Section D.3.3](#) for qualitative traces illustrating these dynamics.

Table 6.5: Tokens per problem (T/P, ↓) at an 8192-token context window. We report total generated tokens per evaluation instance for Think-Answer (TA), the RL-only interleaved baseline of Xie et al. [305], and our interleaved inference strategies. Lower is better; best per column within each model in **bold**.

Method	MATH500	MBPP	Text-to-SQL	QualITY
QWEN3-4B				
TA (8192)	3968.2	2816.8	3052.1	2870.2
Plan-Answer	2348.0	2328.6	1902.4	1850.2
R&R (Plan, 8192)	2496.2	2258.4	2068.3	1814.5
Best-of-N (5, 8192)	2834.6	3022.8	2312.4	2684.2
QWEN3-8B				
TA (8192)	4298.9	2986.8	3214.6	3025.6
Interleaved RL [305]	3010.0	2460.0	2380.0	2290.0
Plan-Answer	2654.6	2489.4	1806.2	1955.8
R&R (Plan, 8192)	2864.1	2492.6	2268.9	1908.2
Best-of-N (5, 8192)	3122.9	3115.4	2428.7	2698.6

Interleaved decoding uses far fewer output tokens per problem than Think-Answer at the same 8192 context. [Table 6.5](#) reports tokens-per-problem (T/P) for TA, the RL-only interleaved baseline of Xie et al. [305], and our strategies; Plan-Answer and R&R (Plan) cut T/P by large margins versus TA (e.g., QWEN3-4B on MATH500 3968 → 2348 for Plan-Answer; Text-to-SQL 3052 → 1902). Best-of-N (5) increases T/P relative to single-plan methods because it scores multiple candidate plans before continuing. Across domains, plan-first generation stays closer to the most token-efficient operating points while preserving the accuracy gains discussed above.

A small number of plan-level interventions recover most failure cases, highlighting the

Table 6.6: Rewind statistics. Fraction of prompts requiring plan rewinds and corresponding approval rates. On average, only 22% of plans require a rewind, and over 80% of corrections succeed within two attempts, demonstrating the efficiency of plan-level R&R.

Metric	MATH500	MBPP	Text-to-SQL	QuaLITY	Average
Plans needing rewind	120/500	55/257	60/250	40/250	22.4%
First rewind approved	60.8%	54.5%	63.3%	67.5%	61.5%
Second rewind approved	25.8%	18.2%	21.7%	17.5%	20.8%
Rewound P@1	86.6%	72.7%	85.0%	82.0%	81.6%

efficiency of structured feedback. Table 6.6 analyzes how often plan-level intervention is needed and how quickly the model converges under the Rewind & Repeat (R&R) strategy. Across all benchmarks, only 20–25% of plans require a rewind, and the majority of those are corrected on the first retry (60–65%). A small fraction (15–25%) benefit from a second rewind, after which success rates exceed 80%. These results indicate that most initial plans are already well-formed and R&R primarily serves to re-ground reasoning early, preventing the model from pursuing unproductive solution paths. This mechanism contributes directly to the token efficiency observed in Table 6.3, where plan-level feedback reduces reasoning depth without sacrificing accuracy. Qualitatively, we find that rewinds promote more stable and targeted reasoning trajectories, allowing the model to converge faster on valid solutions with minimal additional tokens.

6.6 Related Work

LLM Overthinking and Adaptive Thinking. Reasoning models such as OpenAI O1 [136] and DeepSeekR1 [103] are trained using RL-based methods to improve task accuracy on complex logic and reasoning tasks. Such LLMs notoriously suffer from an overthinking phenomenon where they produce verbose and redundant reasoning steps [51, 271], which wastes time, risks exceeding the token budget, and in the extreme, can introduce logical fallacies that lead to incorrect final answers [271].

One proposed solution is to incorporate a thought length penalty into the RL objective, penalizing verbose and incorrect answers while encouraging the model to produce more concise

reasoning steps [8, 204, 321]. Another strategy focuses on data curation, where variable length Chain-of-Thought (CoT) reasoning datasets are constructed for Supervised Fine-tuning (SFT) [144]. This is achieved by using heuristics or another LLM to summarize and compress longer reasoning chains without losing key information. Finally, Munkhbat et al. [221] use best-of-N sampling to generate more concise reasoning paths for data curation. A key difference from all of these approaches is that we do not wish to penalize long reasoning chains *per se*, but rather to improve interactivity and ensure that reasoning effort is well spent. We show that training for interleaved reasoning naturally encourages concise reasoning without explicit length penalties, while allowing users to ensure that reasoning proceeds from the correct premises.

Post-hoc Reranking and Inference-Time Steering. We consider inference-time techniques that exploit plan proposals to improve generation quality. This is related to the broader literature on inference-time scaling and aggregation, which is too large to properly survey here. We draw particular inspiration from early approaches, such as contrastive decoding [183] and energy-based reranking models [18], which generate multiple candidate continuations and select those that best satisfy predefined constraints. RANKGEN [162] introduces a large encoder trained with contrastive objectives to assess prefix–continuation compatibility, improving coherence and topical relevance when applied to candidate selection. RankRAG [327] unifies reranking and answer generation in retrieval-augmented settings. More recently, AGGLM [340] merges reasoning outputs from several trajectories into a unified answer through inference-time aggregation.

While these methods can improve performance, they require producing and filtering many full reasoning traces. In contrast, our inference-time steering techniques are more efficient because they intervene directly within the decoding process itself, by producing and evaluating *partial* generations before continuing with the remainder of the response. Conceptually, this means the unit of generation for evaluation is smaller: instead of scoring entire completions post-hoc, as in RANKGEN or AGGLM, our approach incorporates plan assessment mid-generation. Unlike speculative decoding [178], this does not require interrupting the decoding process or maintaining multiple concurrent hypotheses. The interleaved model naturally alternates between plan-

ning, reasoning, and answering within a single forward pass. This shifts the intervention from post-hoc reranking to real-time guidance, enabling efficiency gains without auxiliary reranking stages.

Plan Guided Reasoning. The ReAct framework interleaves chain-of-thought reasoning with actions allowing the model to incorporate external information sources through tool calls, thereby grounding the model and reducing hallucinations in the final answer [318]. A particularly relevant follow-up is Pre-Act, which creates a multi-step execution plan and reasoning for each action [246]. In contrast to ReAct-style frameworks, PLANTAIN produces intermediate answers that are surfaced to the user allowing them to intervene and correct the model’s reasoning paths. Similar to ReAct, we observe that grounding the reasoning in an explicit plan leads to improved downstream accuracy. Other prior works have shown that plan-based models, sometimes referred to as *blueprints*, help with model robustness and attribution [85, 104, 128].

Interactive LLM Interfaces and Decoding Control. User-facing LLM interfaces, such as OpenAI’s Deep Research, Gemini’s multi-step reasoning mode, and most standard chat UIs allow users to stop generation mid-thought, truncate rambling explanations, or redirect the model before a full chain of thought is completed [232]. While effective for improving perceived latency and user control, UI-level interruption does not influence the model’s internal reasoning policy. As a result, the model continues to plan to overthink, even if the user frequently stops it. From a learning perspective, the system never receives a training signal that shorter or more structured reasoning is preferred. In contrast, interleaved reasoning modifies the generation process itself, prompting the model to surface compact, high-value intermediate outputs by default while continuing internal thought as needed.

6.7 Conclusion

We introduced PLANTAIN, a post-training framework that elicits *interleaved reasoning* behavior in large reasoning models. By encouraging models to first produce an explicit plan and then alternate between internal reasoning and user-facing responses, PLANTAIN reduces latency and

improves controllability. Empirically, interleaved reasoning achieves up to $7\times$ lower time-to-first-response (TTFR) and +2–3% higher task accuracy compared to standard *Think-Answer* decoding, while maintaining comparable total token usage. Our interleaved model learns to structure its reasoning process naturally, without explicit length penalties or handcrafted constraints. At inference time, we introduced two plan-level control strategies, Best-of- N selection and Rewind & Repeat, that further improve response quality and efficiency without additional training. Although trained solely on code generation tasks, the resulting model generalizes effectively to broader reasoning domains such as math and long-context question answering.

PLANTAIN focuses on structured, verifiable tasks where plan correctness can be automatically assessed. An important next step is to conduct human studies evaluating the perceived usefulness, interpretability, and responsiveness of interleaved reasoning in interactive settings. We also plan to extend evaluations to scenarios with ambiguous or under-specified user prompts, where users can provide early feedback on intermediate plans to guide the model toward their true intent. Finally, the LLM-as-a-judge used for plan evaluation introduces nontrivial latency and may reflect biases of the underlying model. Future work could mitigate these effects by incorporating human preference data or lightweight learned reward models to better calibrate plan evaluation and align judgments with user intent.

Impact Statement

This chapter describes fundamental research on techniques for inference-time guidance in reasoning models, specifically through interleaved reasoning where models alternate between internal thought and visible intermediate plans. The primary intended use case that we analyze in this work is interactive problem solving, where reducing perceived latency and enabling user intervention are core challenges. Surfacing intermediate plans can help make AI systems easier to use, correct, and monitor while solving complex tasks in areas such as coding and math. We do not speculate about broader impacts that may follow from this technical contribution.

II

Autonomous Improvement under Distribution Shift

Part II: Autonomous Improvement under Distribution Shift

Part I argued that scalable supervision should draw on cheap, natural signals that people already produce. But no matter how good the supervision is at training time, the world at deployment is not the world the robot was trained on. Lighting changes, object placements drift, user preferences vary, and the long tail of failure modes that a fixed policy must handle cannot be fully anticipated from offline data [10, 240]. A robot that is only as good as its initial training distribution will eventually fail in ways that no amount of additional demonstrations can efficiently fix.

The premise of Part II is that robots must *keep learning* once they leave the training distribution, and that this learning should come from the robot’s own interaction with its environment rather than from ever-larger human-collected datasets. Failures, near-misses, and partial successes during deployment are not noise to be discarded: they are signals that reveal where the policy is weak, what the environment actually contains, and how the task is truly specified. Turning these signals into learning requires two ingredients. First, the system needs a way to structure adaptation so that experience across episodes can inform behavior within the current episode. Second, it needs dense and generalizable feedback that remains meaningful across tasks and embodiments, rather than sparse, task-specific rewards that must be hand-engineered for every new setting.

The two chapters in this part address these ingredients in turn. Chapter 7 develops DynaMITE-

RL, a meta-reinforcement learning framework that models adaptation as part of the learning problem itself, capturing temporal structure across episodes so that a robot can infer and respond to changing task dynamics from its own experience. Chapter 8 introduces Robometer, a video-language reward model that provides dense task progress and success signals from both per-frame supervision and trajectory-level comparisons, giving reinforcement learning and post-training a scalable source of feedback that transfers across robots, scenes, and tasks.

The two chapters in Part II address the deployment-time half of the problem that Part I leaves open. Scalable supervision provides an initial policy; structured adaptation and generalizable reward signals allow that policy to continue improving from its own interaction data. Making robots competent in open-world settings requires progress on both learning algorithms and feedback signals: cheap, natural supervision to initialize a useful policy, and autonomous adaptation to correct the errors that training-time data cannot anticipate.

CHAPTER 7

Chapter 7

DynaMITE-RL: A Dynamic Model for Improved Temporal Meta-Reinforcement Learning

7.1 Introduction

Markov decision processes (MDPs) [28] provide a general framework in reinforcement learning (RL), and can be used to model sequential decision problems in a variety of domains, e.g., recommender systems (RSs), robot and autonomous vehicle control, and healthcare [30, 46, 131, 140, 200, 323]. MDPs assume a static environment with fixed transition probabilities and rewards [24]. In many real-world systems, however, the dynamics of the environment are intrinsically tied to latent factors subject to temporal variation. While nonstationary MDPs are special instances of partially observable MDPs (POMDPs) [143], in many applications these latent variables change infrequently, i.e. the latent variable remains fixed for some duration before changing. One class of problems exhibiting this latent transition structure is recommender systems, where a user's preferences are a latent variable which gradually evolves over time [141, 151]. For instance, a user may initially have a strong affinity for a particular genre (e.g., action movies), but their viewing habits could change over time, influenced by external factors such as trending movies, mood, etc.

A robust system should adapt to these evolving tastes to provide suitable recommendations. Another example is in manufacturing settings, where industrial robots may experience unobserved gradual deterioration of their mechanical components affecting the overall functionality of the system. Accurately modelling such latent transitions caused by hardware degradation can help manufacturers optimize performance, cost, and equipment lifespan.

Our goal in this work is to leverage such a temporal structure to obviate the need to solve a fully general POMDP. To this end, we propose **Dynamic Model for Improved Temporal Meta Reinforcement Learning** (DynaMITE-RL), a method designed to exploit the temporal structure of sessions, i.e., sub-trajectories within the history of observations in which the latent state is fixed. We formulate our problem as a *dynamic latent contextual MDP (DLCMDP)*, and identify three crucial elements needed to enable tractable and efficient policy learning in environments with the latent dynamics captured by a DLCMDP. First, we consider consistency of latent information, by exploiting time steps for which we have high confidence that the latent variable is constant. To do so, we introduce a consistency loss to regularize the posterior update model, providing better posterior estimates of the latent variable. Second, we enforce the posterior update model to learn the dynamics of the latent variable. This allows the trained policy to better infer, and adapt to, temporal shifts in latent context in unknown environments. Finally, we show that the variational objective in meta-RL algorithms, which attempts to reconstruct the entire trajectory, can hurt performance when the latent context is nonstationary. We modify this objective to reconstruct only the transitions that share the same latent context.

Closest to our work is VariBAD [349], a meta-RL [20] approach for learning a Bayes-optimal policy, enabling an agent to quickly adapt to a new environment with unknown dynamics and reward functions. VariBAD uses variational inference to learn a posterior update model that approximates the belief over the distribution of transition and reward functions. It augments the state space with this belief to represent the agent’s uncertainty during decision-making. Nevertheless, VariBAD and the Bayes-Adaptive MDP framework [249] assume the latent context is static *across an episode* and do not address settings with latent state dynamics. In this work, we

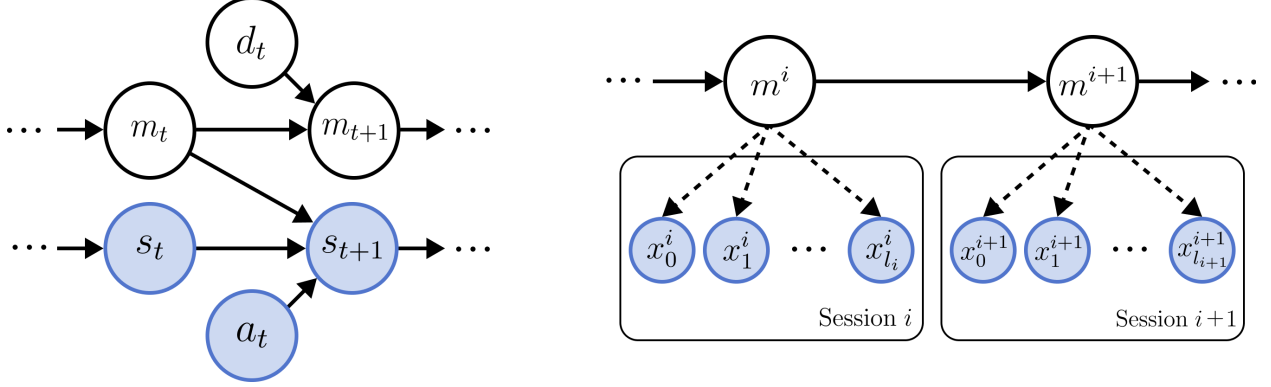


Figure 7.1: **(Left)** The graphical model for a DLCMDP. The transition dynamics of the environment follows $T(s_{t+1}, m_{t+1} \mid s_t, a_t, m_t)$. At every timestep t , an i.i.d. Bernoulli random variable, d_t , denotes the change in the latent context, m_t . Blue shaded variables are observed and white shaded variables are latent. **(Right)** A DLCMDP rollout. Each session i is governed by a latent variable m^i which is changing between sessions according to a fixed transition function, $T_m(m' \mid m)$. We denote l_i as the length of session i . The state-action pair (s_t^i, a_t^i) at timestep t in session i is summarized into a single observed variable, x_t^i . We emphasize that session terminations are not explicitly observed.

focus on the dynamic latent state formulation of the meta-RL problem.

Our core contributions are as follows: (1) We introduce DynaMITE-RL, a meta-RL approach to handle environments with evolving latent context variables. (2) We introduce three key elements for learning an improved posterior update model: session consistency, modeling dynamics of latent context, and session reconstruction masking. (3) We validate our approach on a diverse set of challenging simulation environments and demonstrate significantly improved results over multiple state-of-the-art baselines in both online and offline-RL settings.

7.2 Background

We begin by reviewing relevant background including meta-RL and Bayesian RL. We also briefly summarize the VariBAD [349] algorithm for learning Bayes-Adaptive policies.

Meta-RL. The goal of meta-RL [20] is to quickly adapt an RL agent to an unseen test environment. Meta-RL assumes a distribution $p(\mathcal{T})$ over possible environments or *tasks*, and learns this distribution by repeatedly sampling batches of tasks during meta-training. Each task $\mathcal{T}_i \sim p(\mathcal{T})$ is described by an MDP $\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, R_i, T_i, \gamma)$, where the state space $\mathcal{S}$, action space $\mathcal{A}$, and discount factor γ are shared across tasks, while R_i and T_i are task-specific reward and transition

functions, respectively. The objective of meta-RL is to learn a policy that efficiently maximizes reward given a new task $\mathcal{T}_i \sim p(\mathcal{T})$ sampled from the task distribution at meta-test time. Meta-RL is a special case of a POMDP in which the unobserved variables are R and T , which are assumed to be stationary throughout an episode.

Bayesian Reinforcement Learning (BRL). BRL [98] utilizes Bayesian inference to model the uncertainty of agent and environment in sequential decision making problems. In BRL, R and T are unknown a priori and treated as random variables with associated prior distributions. At time t , the *observed history* of states, actions and rewards is $\tau_{:t} = \{s_0, a_0, r_1, \dots, r_t, s_t\}$, and the belief b_t represents the posterior over task parameters R and T given the transition history, i.e. $b_t \triangleq p(R, T \mid \tau_{:t})$. Given the initial belief $b_0(R, T)$, the belief can be updated iteratively using Bayes’ rule: $b_{t+1} = p(R, T \mid \tau_{:t+1}) \propto p(s_{t+1}, r_{t+1} \mid \tau_{:t}, R, T) \cdot b_t$. This Bayesian approach to RL can be formalized as a *Bayes-*

Adaptive MDP (BAMDP) [79]. A BAMDP is an MDP over the *augmented state space* $S^+ = S \times \mathcal{B}$, where $\mathcal{B}$ denotes the belief space. Given the augmented state $s_t^+ = (s_t, b_t)$, the transition function is given by $T^+(s_{t+1}^+ \mid s_t^+, a_t) = \mathbb{E}_{b_t} [T(s_{t+1} \mid s_t, a_t) \cdot \delta(b_{t+1} = p(R, T \mid \tau_{:t+1}))]$, and reward function under the current belief is, $R^+(s_t^+, a_t) = \mathbb{E}_{b_t} [R(s_t, a_t)]$. The BAMDP formulation naturally resolves the exploration-exploitation tradeoff. A Bayes-optimal RL agent takes information-gathering actions to reduce its uncertainty in the MDP parameters while simultaneously maximizing the task returns. However, for most interesting problems, solving the BAMDP—and even computing posterior updates—is intractable given the continuous and typically high-dimensional nature of the task distribution.

VariBAD. Zintgraf et al. [349] approximates the Bayes-optimal solution by modeling uncer-

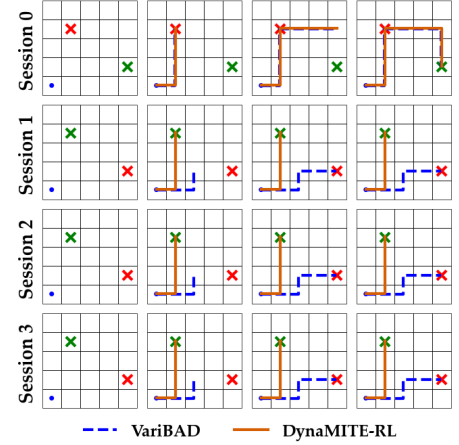


Figure 7.2: VariBAD does not model the latent context dynamics and fails to adapt to the changing goal location. By contrast, DynaMITE-RL correctly infers the transition and consistently reaches the rewarding cell (green cross).

tainty over the MDP parameters. These parameters are represented by a latent vector $m \in \mathbb{R}^d$, the posterior over which is $p(m \mid \tau_{:H})$, where H is the BAMDP horizon. VariBAD uses a variational approximation $q_\phi(m \mid \tau_{:t})$ parameterized by ϕ and conditioned on the observed history up to time t . Zintgraf et al. [349] show that $q_\phi(m \mid \tau_{:t})$ approximates the belief b_t . In practice, $q_\phi(m \mid \tau_{:t})$ is represented by a Gaussian distribution $q_\phi(m \mid \tau_{:t}) = \mathcal{N}(\mu(\tau_{:t}), \Sigma(\tau_{:t}))$, where μ and Σ are sequence models (e.g., recurrent neural networks or transformers [288]) that encode trajectories to latent statistics. The variational lower bound at time t is $\mathbb{E}_{q_\phi(m \mid \tau_{:t})}[\log p_\theta(\tau_{:H} \mid m)] - D_{KL}(q_\phi(m \mid \tau_{:t}) \parallel p_\theta(m))$, where the first term reconstructs the trajectory likelihood $p_\theta(\tau_{:H} \mid m)$ and the second term regularizes the variational posterior to a prior distribution over the latent space, typically modeled with a standard Gaussian distribution. Importantly, the trajectory up to time t , i.e., $\tau_{:t}$, is used in the ELBO equation to infer the posterior belief at time t , which then decodes the entire trajectory $\tau_{:H}$, *including future transitions*. Given the belief state distribution q_ϕ of a BAMDP, the policy maps both the state and belief to actions, i.e., $\pi(a_t \mid s_t, q_\phi(m \mid \tau_{:t}))$. The BAMDP solution policy π^* is trained, e.g., via policy gradient methods, to maximize the expected cumulative return over the task distribution: $J(\pi) = \mathbb{E}_{R,T} [\mathbb{E}_\pi [\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)]]$.

7.3 Dynamic Latent Contextual MDPs

As a special case of a BAMDP, where the belief state is parameterized with a latent context vector (analogous to the problem formulation of VariBAD), the *dynamic latent contextual MDP (DL-CMDP)* is denoted by $\langle \mathcal{S}, \mathcal{A}, \mathcal{M}, R, T, \nu_0, H \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{M}$ is the *latent* context space, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{M} \mapsto \Delta_{[0,1]}$ is a reward function, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{M} \mapsto \Delta_{\mathcal{S} \times \mathcal{M}}$ is a transition function, $\nu_0 \in \Delta_{\mathcal{S} \times \mathcal{M}}$ is an initial state distribution, $\gamma \in (0, 1)$ is a discount factor, and H is the (possibly infinite) horizon.

We assume an episodic setting in which each episode begins in a state-context pair $(s_0, m_0) \sim \nu_0$. At time t , the agent is at state s_t and context m_t , and has observed history $\tau_{:t} = \{s_0, a_0, r_1, \dots, r_t, s_t\}$. Given the history, the agent selects an action $a_t \in \mathcal{A}$, after which the state and latent context transitions according to $T(s_{t+1}, m_{t+1} \mid s_t, a_t, m_t)$, and the agent receives a reward sampled from

$R(s_t, a_t, m_t)$. Throughout this process, the context m_t is latent (i.e., *not observed* by the agent).

DLCMDPs embody the causal independence depicted by the graphical model in Figure 7.1. Particularly, DLCMDPs impose a structure on changes of the latent variable m , allowing the latent context m to change less or more frequently. We denote by d_t the random variable at which a transition occurs in m_t . According to Figure 7.1, the transition function T is represented by the following factored distribution:

$$\begin{aligned} T(s_{t+1} = s', m_{t+1} = m' \mid s_t = s, a_t = a, m_t = m) \\ = T_s(s' \mid s, a, m) \mathbb{1} \{m' = m, d_t = 0\} T_d(d_t = 0) + \nu_0(s' \mid m') T_m(m' \mid m) \mathbb{1} \{d_t = 1\} T_d(d_t = 1), \end{aligned}$$

where $T_m : \mathcal{M} \mapsto \mathcal{M}$ is the latent dynamics function, T_s is the context-dependent state transition function, and T_d is the termination probability distribution. We refer to sub-trajectories between changes in the latent context as *sessions*, which may vary in length. At the start of a new episode, a new state and a new latent context are sampled based on the distribution ν_0 . Each session itself is governed by an MDP parameterized with a latent context $m \in \mathcal{M}$, which changes stochastically between sessions according to the latent transition function $T_m(m' \mid m)$. For notational simplicity we use index i to denote the i^{th} session in a trajectory, and m^i the respective latent context of that session. We emphasize that sessions switching times are latent random variables.

Notice that DLCMDPs are more general than latent MDPs [167, 269], in which the latent context is fixed throughout the entire episode; this corresponds to $d_t \equiv 0$. Moreover, DLCMDPs are closely related to POMDPs; letting $d_t \equiv 1$, a DLCMDP reduces to a general POMDP with state space $\mathcal{M}$, observation space $\mathcal{S}$, and observation function ν_0 . As a consequence DLCMDPs are as general as POMDPs, rendering them very expressive. Moreover, the specific temporal structure of DLCMDPs allows us to devise efficient learning algorithms that exploit the transition dynamics of the latent context, improving learning efficiency. DLCMDPs are related to DCMDPs [280], LSMDPs [49], and DP-MDP [303]. However, DCMDPs assume contexts are observed, and focus on aggregated context dynamics, LSMDPs assume that the latent contexts across sessions are i.i.d (i.e., there is no latent dynamics) and DP-MDPs assume that sessions are fixed length.

We aim to learn a policy $\pi(a_t \mid s_t, m_t)$ which maximizes the expected return $J(\pi)$ over un-

Algorithm 8 DynaMITE-RL Training

- 1: **Input:** env, policy, critic, belief model
 - 2: **for** iter = 1 to num_rl_updates **do**
 - 3: Collect DLCMDP episodes
 - 4: Train posterior belief model by maximizing ELBO (Eq. (7.4.2))
 - 5: Train policy and critic with any online RL algorithm
-

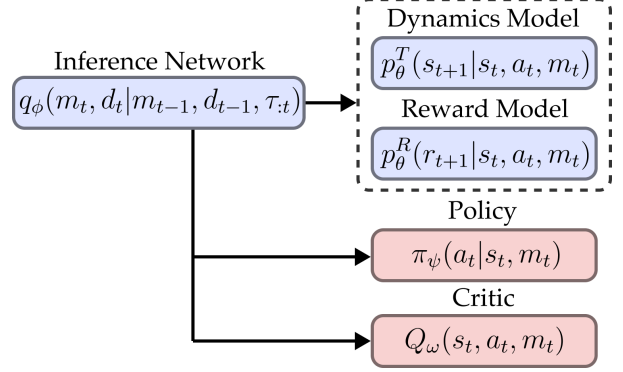


Figure 7.3: Pseudo-code (online RL training) and model architecture of DynaMITE-RL.

seen test environments. As in BAMDPs, the optimal DLCMDP Q -function satisfies the Bellman equation; $\forall s^+ \in S^+, a \in \mathcal{A} : Q(s^+, a) = R^+(s^+, a) + \gamma \sum_{s'^+ \in S^+} T^+(s'^+ | s^+, a) \max_{a'} Q(s'^+, a)$. In the following section, we present DynaMITE-RL for learning a Bayes-optimal agent in a DLCMDP.

7.4 DynaMITE-RL

We detail DynaMITE-RL, first deriving a variational lower bound for learning a DLCMDP posterior model, then outlining three principles for training DLCMDPs, and finally integrating them into our training objective.

Variational Inference for Dynamic Latent Contexts. Given that we do not have direct access to the transition and reward functions of the DLCMDP, following Zintgraf et al. [349], we infer the posterior $p(m | \tau_{:t})$, and reason about the latent context vector m instead. Since exact posterior computation over m is computationally infeasible, given the need to marginalize over task space, we introduce the variational posterior $q_\phi(m | \tau_{:t})$, parameterized by $\phi \in \mathbb{R}^d$, to enable fast inference at every step. Our learning objective maximizes the log-likelihood $\mathbb{E}_\pi[\log p(\tau)]$ of observed trajectories. In general, the true posterior over the latent context is intractable, as is the empirical estimate of the log-likelihood. To circumvent this, we derive the *evidence lower bound* (ELBO) [156] to approximate the posterior over m under the variational inference framework.

Let $\mathcal{Z} = \{m^i\}_{i=0}^{K-1}$ be the sequence of latent context vectors for K sessions in an episode (note that K is inherently a random variable—the exact number of sessions in an episode is not

known) and $\Omega = \{d_t\}_{t=0}^{H-1}$ denote the collection of session terminations. We use a parametric generative distribution model for the state-reward trajectory, conditioned on the action sequence: $p_\theta(s_0, r_1, s_1, \dots, r_H, s_H \mid a_0, \dots, a_{H-1})$. In what follows, we drop the conditioning on $a_{:H-1}$ for the sake of brevity.

The variational lower bound can be expressed as:

$$\log p_\theta(\tau) \geq \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} [\log p_\theta(\tau \mid \mathcal{Z}, \Omega)]}_{\text{trajectory reconstruction}} - \underbrace{D_{KL}(q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t}) \parallel p_\theta(\mathcal{Z}, \Omega))}_{\text{prior regularization}} = \mathcal{L}_{\text{ELBO}, t}, \quad (7.4.1)$$

which can be estimated via Monte Carlo sampling over a learnable approximate posterior q_ϕ . In optimizing the reconstruction loss of session transitions and rewards, the learned latent variables should capture the unobserved MDP parameters. The full derivation of the ELBO for a DLCMDP is provided in Appendix E.1.

Figure 7.2 depicts a (qualitative) didactic GridWorld example with two possible rewarding goals that alternate between sessions. The VariBAD agent does not account for latent goal dynamics and gets stuck after reaching the goal in the first session. By contrast, DynaMITE-RL employs the latent context dynamics model to capture goal changes, and adapts to the context changes across sessions.

Consistency of Latent Information. In the DLCMDP formulation, each session is itself an MDP with a latent context fixed across the session. This within-context stationarity means new observations can only increase the information the agent has about this context. In other words, the agent’s posterior over latent contexts should gradually hone in on the true latent distribution. Although this true distribution remain unknown, this insight suggest the use of a *session-based consistency loss*, which penalizes the agent if there is no increase in information between timestep. Our consistency objective penalizes the agent when the difference between KL-divergence of the posterior to the final posterior in the session between consecutive timesteps is positive, which is the case when there is no increase in information about a session’s latent context after observing a new transition. Let $d_{H-1} = 1$ and $t_i \in \{0, \dots, H\}$ be a random variable denoting the last timestep of session $i \in \{0, \dots, K-1\}$, i.e., $t_i = \min\{t' \in \mathbb{Z}_{\geq 0} : \sum_{t=0}^{t'} d_t = i + 1\}$. For time t in session i , we

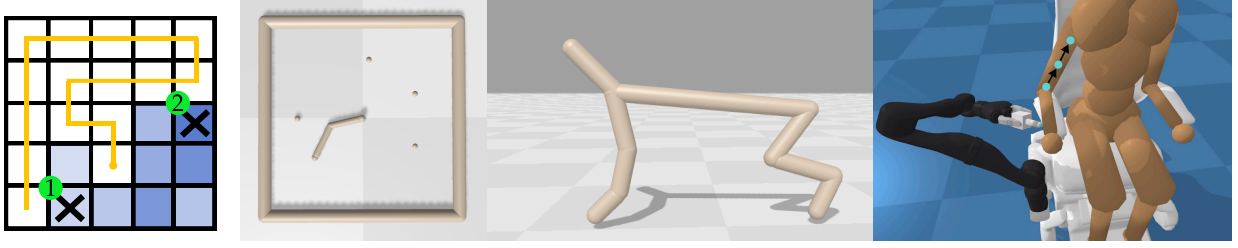


Figure 7.4: The environments considered in evaluating DynaMITE-RL. Each environment exhibits some change in reward and/or dynamics between sessions including changing goal locations (left and middle left), changing target velocities (middle right), and evolving user preferences of itch location (right).

define,

$$\delta_t = D_{KL}(q_\phi(m^i | \tau_{:t+1}) \parallel q_\phi(m^i | \tau_{:t_i})),$$

where $q_\phi(m^i | \tau_{:t_i})$ is the final posterior in session i . This measures the difference between our current belief at time t to the final belief at the end of the session. Our temporal, session-based consistency objective is

$$\mathcal{L}_{\text{consistency},t} = \max\{\delta_{t+1} - \delta_t, 0\}.$$

Using temporal consistency to regularize inference introduces an explicit inductive bias that allows for better posterior estimation.

Remark 7.4.1. We introduce session-based consistency for DLCMDPs, though it is also relevant in single-session settings with stationary latent context. Indeed, as we discuss below, while VariBAD focuses on single sessions, it does not constrain the latent’s posterior to be identical to final posterior belief. Consistency may be useful in settings where the underlying latent variable is stationary, but may hurt performance when this variable is indeed changing. Since our modeling approach allows latent context changes across sessions, incorporating consistency regularization does not generally hurt performance.

Latent Belief Conditioning. Unlike the usual BAMDP framework, DLCMDPs allow one to model temporal changes of latent contexts via dynamics $T_m(m' | m)$ across sessions. To incorporate this model into belief estimation, in addition to the history $(\tau_{:t}, d_{:t})$, we condition the posterior on the final latent belief $q_\phi(m', d' | m, d, \tau_{:t})$ from the previous session, and impose

KL-divergence matching between this belief and the prior distribution $p_\theta(m' | m)$.

Reconstruction Masking. When the agent is at time t , Zintgraf et al. [349] encodes past interactions to obtain the current posterior $q_\phi(m | \tau_{:t})$ since this is all the information available for inference about the current task (see Eq. (7.4.1)). They use this posterior to decode the entire trajectory—including *future transitions*—from different sessions to optimize the lower bound during training. The insight is that decoding both the past and future allows the posterior model to perform inference about unseen states. However, we observe that when the latent context is stochastic, reconstruction over the full sequence is detrimental to training efficiency. The model is attempting to reconstruct transitions outside of the current session that may be irrelevant or biased given the latent state dynamics, rendering it a more difficult learning problem. Instead we reconstruct only the transitions within the session defined by the predicted termination indicators, i.e., at any arbitrary time t within session i , the session-based reconstruction loss is given by

$$\mathcal{L}_{\text{session-ELBO},t} = \mathbb{E}_{q_\phi(\mathcal{Z}, \Omega | \tau_{:t})} \left[\log p_\theta(\tau_{t_{i-1}+1:t_i} | \mathcal{Z}, \Omega) \right] - D_{KL}(q_\phi(\mathcal{Z}, \Omega | \tau_{:t}) \| p_\theta(\mathcal{Z}, \Omega)),$$

where t_i is the last timestep of session i .

DynaMITE-RL. By incorporating the three modifications above, we obtain at the following training objective for our variational meta-RL approach:

$$\mathcal{L}_{\text{DynaMITE-RL}}(\theta, \phi) = \sum_{t=0}^{H-1} \left[\mathcal{L}_{\text{session-ELBO},t}(\theta, \phi) + \beta \mathcal{L}_{\text{consistency},t}(\phi) \right], \quad (7.4.2)$$

where $\beta > 0$ is a hyperparameter that balances the consistency loss with the ELBO objective.

We present a simplified pseudocode for online training of DynaMITE-RL in Algorithm 8 and a detailed algorithm in Appendix E.2.

Implementation Details. We use Proximal Policy Optimization (PPO) [257] for online RL training. We introduce a posterior inference network that outputs a Gaussian over the latent context for the i -th session and the session termination indicators, $q_\phi(m_{t+1}, d_{t+1} | \tau_{:t}, m_t, d_t)$, conditioned on the history and posterior belief from the previous session. We parameterize the

inference network as a sequence model, with e.g., an RNN [54] or a Transformer [288], with different multi-layer perceptron (MLP) output heads for predicting the logits for session termination and the posterior belief. In practice, the posterior belief MLP outputs the parameters of a Gaussian distribution $q_{\phi_m}(m_{t+1} \mid \tau_{:t}, m_t) = \mathcal{N}(\mu(\tau_{:t}), \Sigma(\tau_{:t}))$ where the variance represents the agent’s uncertainty about the MDP. The session termination network applies a sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ to the MLP output. Following PPO [257], the actor loss $\mathcal{J}_\pi$ and critic loss $\mathcal{J}_\omega$ are respectively given by $\mathcal{J}_\pi = \mathbb{E}_{\tau \sim \pi_\psi} [\log \pi_\psi(a \mid s, m) A(s, a, m)]$ and $\mathcal{J}_\omega = \mathbb{E}_{\tau \sim \pi_\psi} [(Q_\omega(s, a, m) - (r + V_\omega(s', m)))^2]$, where V is the state-value network, Q is the state-action value network, and A is the advantage function. We also add an entropy bonus to ensure sufficient exploration in more complex domains. A decoder network, also parameterized using MLPs, reconstructs transitions and rewards given the session’s latent context m , current state s_t , and action a_t , i.e., $p_\theta^T(s_{t+1} \mid s_t, a_t, m_t)$ and $p_\theta^R(r_{t+1} \mid s_t, a_t, m_t)$. Figure 7.3 depicts the implemented model architecture. The final objective is to jointly learn the policy π_ψ , the variational posterior model q_ϕ , and the factored likelihood model p_θ that minimizes the following loss:

$$\mathcal{L}(\theta, \phi, \psi) = \mathbb{E} \left[\mathcal{J}_\pi(\psi) + \lambda \cdot \mathcal{L}_{\text{DynaMITE-RL}}(\phi, \theta) \right], \quad (7.4.3)$$

where $\mathcal{J}$ is the expected return, and $\lambda > 0$ is a hyperparameter balancing the RL objective with DynaMITE-RL’s variational inference objective. We also evaluate DynaMITE-RL in an offline RL setting, in which we collect an offline dataset of trajectories following an oracle goal-conditioned policy and subsequently approximate the optimal value function and RL agent using offline RL methods, e.g., IQL [160]. The value function and the policy are parameterized with the same architecture as in the online setting and will be detailed in Appendix E.5.

7.5 Experiments

We present experiments that demonstrate, while VariBAD and other meta-RL methods struggle to learn good policies given nonstationary latent contexts, DynaMITE-RL exploits the causal structure of a DLCMDP to more efficiently learn performant policies. We compare our approach to several state-of-the-art meta-RL baselines, showing significantly better evaluation returns.

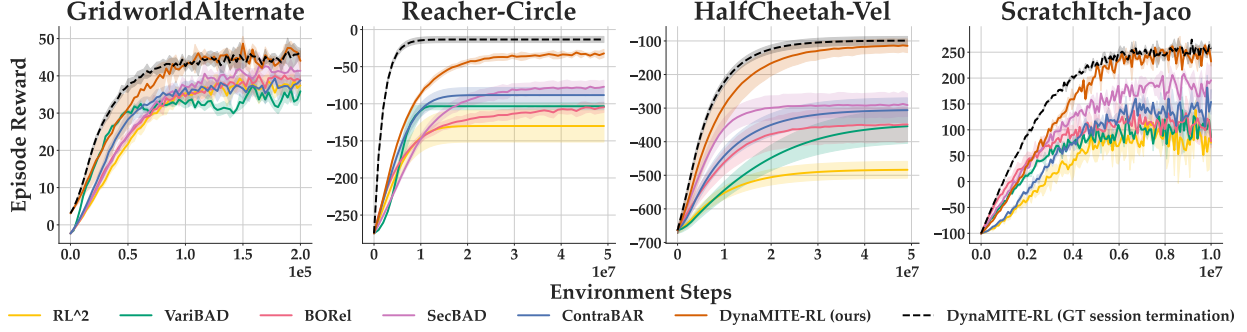


Figure 7.5: Learning curves for **DynaMITE-RL** and state-of-the-art baseline methods. Shaded areas represent standard deviation over 5 different random seeds for each method and 3 for ScratchItch. In each of the evaluation environments, we observe that **DynaMITE-RL** exhibits better sample efficiency and converges to a policy with better environment returns than the baseline methods.

Environments. We test DynaMITE-RL on a suite of standard meta-RL benchmark tasks including a didactic gridworld navigation, continuous control, and human-in-the-loop robot assistance as shown in Figure 7.4. Gridworld navigation and MuJoCo [284] locomotion tasks are considered by Zintgraf et al. [349], Dorfman et al. [72], and Choshen and Tamar [55]. We modify these environments to incorporate temporal shifts in the reward function and/or environment dynamics. To achieve good performance under these conditions, a learned policy must adapt to the latent state dynamics. More details about the environments and hyperparameters can be found in Appendix E.4 and E.5.

Gridworld. We modify the Gridworld environment used by Zintgraf et al. [349]. In a 5×5 gridworld, two possible goals are sampled uniformly at random in each episode. One of the two goals has a +1 reward while the other has 0 reward. The rewarding goal location changes after each session according to a predefined transition function. Goal locations are provided to the agent in the state—the only latent information is which goal has positive reward.

Continuous Control. We experiment with two tasks from OpenAI Gym [39]: Reacher and HalfCheetah. Reacher is a two-jointed robot arm tasked with reaching a 2D goal location that moves along a circular path according to some unknown transition function. HalfCheetah is a locomotion task which we modify to incorporate changing latent contexts w.r.t. the target direction (HalfCheetah-Dir), target velocity (HalfCheetah-Vel), and target velocity with opposing

wind forces (HalfCheetah-Wind+Vel).

Assistive Itch Scratching. Assistive Itch Scratch is part of the Assistive-Gym benchmark [80] consisting of a human and a wheelchair-mounted 7-degree-of-freedom (DOF) Jaco robot arm. The human has limited-mobility and requires robot assistance to scratch an itch. We simulate stochastic latent context by moving the itch location—unobserved by the agent—along the human’s right arm.

Meta-RL Baselines. We compare DynaMITE-RL to several state-of-the-art (approximately) Bayes-optimal meta-RL methods including RL^2 [78], VariBAD [349], BOREl [72], SecBAD [49], and ContraBAR [55]. RL^2 [78] is an RNN-based policy gradient method which encodes environment transitions in the hidden state and maintains them across episodes. VariBAD reduces to RL^2 without the decoder and the variational reconstruction objective for environment transitions. BOREl primarily investigates offline meta-RL (OMRL) and proposes a few modifications such as reward relabelling to address the identifiability issue in OMRL. We evaluate the off-policy variant of BOREl, trained using Soft-Actor Critic (SAC) in our DLCMDP environments. Chen et al. [49] proposes the latent situational MDP (LS-MDP), in which there is non-stationary latent contexts that are sampled i.i.d., and SecBAD, an algorithm for learning in an LS-MDP. However, they do not consider latent dynamics which a crucial aspect in many applications. ContraBAR employs a contrastive learning objective to discriminate future observations from negative samples to learn an *approximate* sufficient statistic of the history. As Zintgraf et al. [349] already demonstrate better performance by VariBAD than posterior sampling methods (e.g., PEARL [245]) we exclude such methods from our comparison.

DynaMITE-RL outperforms prior meta-RL methods in a DLCMDP in both online and offline RL settings. In Figure 7.5, we show the learning curves for DynaMITE-RL and baseline methods. We first observe that DynaMITE-RL significantly outperforms the baselines across all domains in sample efficiency and average environment returns.

RL^2 , VariBAD, BOREl, SecBAD, and ContraBAR all perform poorly in the DLCMDP, converging to

Table 7.1: Average single episode returns for **DynaMITE-RL** and other state-of-the-art meta-RL algorithms across different environments. Results for all environments are averaged across 5 seeds beside ScratchItch which has 3 seeds. **DynaMITE-RL**, in bold, achieves the highest return on all of the evaluation environments and is the only method able to recover an optimal policy.

	Gridworld	Reacher	HC-Dir	HC-Vel	Wind+Vel	ScratchItch
RL ²	33.4 $\pm$ 1.6	-150.6 $\pm$ 1.2	-420.0 $\pm$ 8.4	-513.2 $\pm$ 8.7	-493.5 $\pm$ 1.8	50.4 $\pm$ 16.8
VariBAD	31.8 $\pm$ 1.9	-102.4 $\pm$ 4.2	-242.5 $\pm$ 4.8	-363.5 $\pm$ 3.2	-188.5 $\pm$ 4.4	81.8 $\pm$ 6.9
BORel	32.4 $\pm$ 2.4	-103.5 $\pm$ 4.6	-240.6 $\pm$ 4.3	-343.4 $\pm$ 3.6	-167.8 $\pm$ 5.4	82.5 $\pm$ 6.0
SecBAD	38.5 $\pm$ 3.1	-96.2 $\pm$ 4.8	-202.4 $\pm$ 10.4	-323.5 $\pm$ 3.4	-155.3 $\pm$ 5.4	101.4 $\pm$ 9.2
ContraBAR	34.5 $\pm$ 0.9	-101.6 $\pm$ 3.2	-256.5 $\pm$ 3.6	-312.3 $\pm$ 4.8	-243.4 $\pm$ 2.6	114.6 $\pm$ 24.4
DynaMITE-RL	42.9 $\pm$ 0.5	-8.4 $\pm$ 5.1	-68.5 $\pm$ 2.3	-146.0 $\pm$ 8.1	-42.8 $\pm$ 6.9	231.2 $\pm$ 23.3

a suboptimal policy. **VariBAD** and **BORel** perform comparably as both share similar architecture, the only difference being the RL algorithm. By contrast, **DynaMITE-RL** accurately models the latent dynamics and consistently achieves high returns despite the nonstationary latent context.

We also evaluate an oracle with access to ground-truth session terminations and find that **DynaMITE-RL** with learned session terminations effectively recovers session boundaries and matches oracle performance with sufficient training. Our empirical results validate that **DynaMITE-RL** learns a policy robust to changing latent contexts at inference time, while the baseline methods fail to adapt and are ultimately stuck in suboptimal behavior. We further demonstrate that **DynaMITE-RL** outperforms **BORel** in an offline RL setting in Table 7.2 across all environments. This high-

lights the importance of **DynaMITE-RL** training objectives in learning a more accurate posterior belief model even without online environment interactions. We also experimented with a Trans-

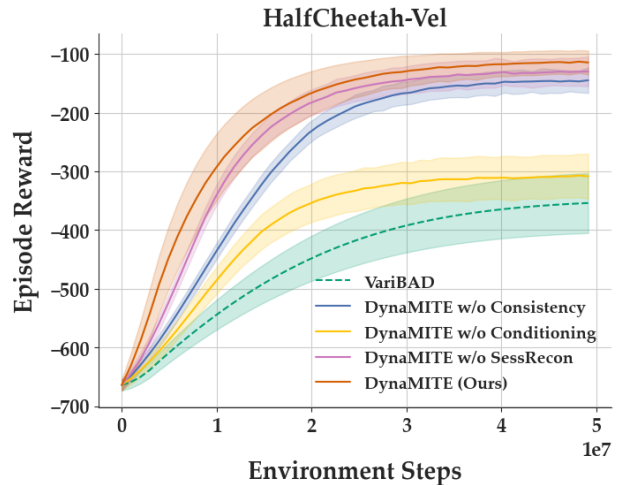


Figure 7.6: Ablating individual components of **DynaMITE-RL**. We observe that modelling latent dynamics is crucial in achieving good performance in a DLCMDP. Additionally, both consistency regularization and session reconstruction are critical for improving the sample efficiency and convergence to a better performing policy.

Table 7.2: Average single episode returns with offline RL. Results are averaged across 5 random seeds. Algorithm with the highest average return are shown in bold. We present results for an oracle agent trained with goal information for reference.

	Gridworld	Reacher	HC-Dir	HC-Vel	HC-Dir+Vel	ScratchItch
BOrel	31.4 $\pm$ 3.5	-102.0 $\pm$ 5.8	-245.0 $\pm$ 12.4	-354.0 $\pm$ 8.3	-170.0 $\pm$ 5.4	72.5 $\pm$ 4.6
w/o Consistency	38.2 $\pm$ 1.2	-33.2 $\pm$ 2.7	-206.0 $\pm$ 5.6	-212.0 $\pm$ 6.4	-120.0 $\pm$ 12.4	105.8 $\pm$ 8.5
w/o Sess. Dynamics	33.4 $\pm$ 1.3	-95.0 $\pm$ 5.2	-244.0 $\pm$ 6.0	-342.0 $\pm$ 8.6	-166.0 $\pm$ 9.5	74.1 $\pm$ 2.3
DynaMITE-RL	41.8 $\pm$ 0.6	-15.5 $\pm$ 3.2	-154.0 $\pm$ 8.6	-156.0 $\pm$ 4.8	-48.0 $\pm$ 8.6	225.5 $\pm$ 10.6
w/ Transformer	43.8 $\pm$ 0.6	-8.4 $\pm$ 2.8	-132.0 $\pm$ 7.4	-144.0 $\pm$ 6.5	-33.0 $\pm$ 5.8	242.5 $\pm$ 7.4
Oracle (w/ goal)	44.6	-4.8	-112.0	-132.2	-24.4	245.3

former encoder to parameterize our belief model and find that a more powerful model further improves the evaluation performance.

Each component of DynaMITE-RL contributes to efficient learning in a DL-CMDP. We ablate the three key components of DynaMITE-RL to understand their impact on the resulting policy. We compare full DynaMITE-RL to: (i) DynaMITE-RL w/o Consistency, which does not include consistency regularization; (ii) DynaMITE-RL w/o Conditioning, which does not include latent conditioning; and (iii) DynaMITE-RL w/o Session Recon, which does not include session reconstruction. In Figure 7.6, we report the performance for each of these ablations and vanilla VariBAD for comparisons. First, without prior latent belief conditioning, the model

converges to a suboptimal policy slightly better than VariBAD, confirming the importance of

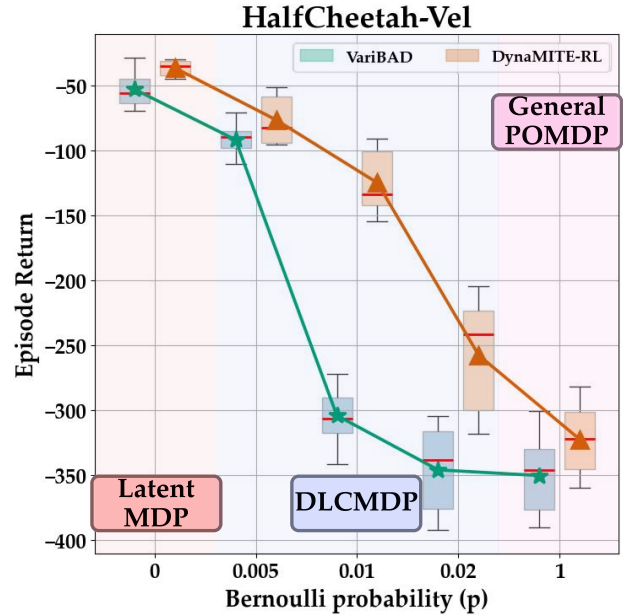


Figure 7.7: Ablation studies on various frequencies of latent context switches within an episode in the HalfCheetah-Vel environment. The boxplot shows the distribution over evaluation returns for 25 rollouts of trained policies with VariBAD and DynaMITE-RL. When $p = 0$, we have a latent MDP and when $p = 1$ this is equivalent to a general POMDP.

modeling the latent transition dynamics of a DLCMDP. Second, we find that session consistency regularization reinforces the inductive bias of changing dynamics and improves the sample efficiency of learning an accurate posterior model in DLCMDPs. Finally, session reconstruction masking also improves the sample efficiency by neglecting terms that are irrelevant and potentially biased. Similar ablation studies in the offline RL setting can be found in Table 7.2, reinforcing the importance of our proposed training objectives.

DynaMITE-RL is robust to varying levels of latent stochasticity. We study the effect of varying the number of latent context switches over an episode of fixed time horizon. For the HalfCheetah-Vel environment, we fix the episode horizon $H = 400$ to create multiple problems. We introduce a Bernoulli random variable, e.g. $d_t \sim \text{Bernoulli}(p)$ where p is a hyperparameter we set to determine the probability that the latent context changes at timestep t . If $p = 0$, the latent context remains unchanged throughout the entire episode, corresponding to a latent MDP. If $p = 1$, the latent context changes at every timestep, which is equivalent to a general POMDP. As shown in Figure 7.7, **DynaMITE-RL** performs better, on average, than **VariBAD**, with lower variance in a latent MDP. We hypothesize that, in the case of latent MDP, consistency regularization helps learn a more accurate posterior model by enforcing the inductive bias that the latent is static. Otherwise, there is no inherent advantage in modeling the latent dynamics if it is stationary.

As we gradually increase the number of context switches, the problem becomes more difficult and closer to a general POMDP. **VariBAD** performance decreases drastically because it is unable to model the changing latent dynamics while **DynaMITE-RL** is less affected, highlighting the robustness of our approach to changing latent contexts. When we set the number of contexts equal to the episode horizon length, we recreate a fully general POMDP and again the performance between **VariBAD** and **DynaMITE-RL** converges.

7.6 Related Work

POMDPs provide a general framework modeling non-stationality and partial observability in sequential decision problems. Many model variants have been introduced, defining a rich spectrum between episodic MDPs and POMDPs. The Bayes-Adaptive MDP (BAMDP) [79] and hidden parameter MDP (HiP-MDP) [150] are both special cases of POMDPs in which environment parameters are unknown and the goal is to infer these parameters online during an episode. The BAMDP model treats unknown parameters as latent variables, which are updated based on the agent’s observations, while the HiP-MDP assumes that the environment dynamics depend on hidden parameters that must be learned over time. However, neither framework addresses the dynamics of the latent parameters across sessions, but rather assumes it is constant throughout an episode.

On the other hand, models like the Latent Situational MDP (LSMDP) [49] and Dynamic Parameter MDP (DP-MDP) [302] do investigate nonstationary latent contexts. LSMDP [49] samples the latent contexts independently and identically distributed (i.i.d.) at each episode. While it introduces variability, it does not model the temporal dynamics or dependencies of these latent parameters. The DP-MDP framework addresses these dynamics by assuming that the latent parameters change at fixed intervals (fixed session lengths), making it less flexible when sessions are variable lengths. By contrast, DLCMDPs models the dynamics of the latent state and simultaneously infers *when* the transition occurs, allowing better posterior updates at inference time.

DynaMITE-RL shares conceptual similarities with other meta-RL algorithms. Firstly, optimization-based techniques [59, 87, 251] learn neural network policies that can quickly adapt to new tasks at test time using policy gradient updates. This is achieved using a two-loop optimization structure: in the inner loop, the agent performs task-specific updates where it fine-tunes the policy with a few gradient steps using the task’s reward function. In the outer loop, the meta-policy parameters are updated based on the performance of these fine-tuned policies across different tasks. However, these methods do not optimize for Bayes-optimal behavior and generally exhibit suboptimal

test-time adaptation. Context-based meta-RL techniques aim to learn policies that directly infer task parameters at test time, conditioning the policy on the posterior belief. Such methods include recurrent memory-based architectures [21, 78, 172, 293] and variational approaches [72, 126, 349]. VariBAD, closest to our work, uses variational inference to approximate Bayes-optimal policies. However, we have demonstrated above the limitations of VariBAD in DLCMDPs, and have developed several crucial modifications to drive effective learning a highly performant policies in our setting.

7.7 Conclusion

We developed DynaMITE-RL, a meta-RL method to approximate Bayes-optimal behavior using a latent variable model. We presented the dynamic latent contextual Markov Decision Process (DLCMDP), a model in which latent context information changes according to an unknown transition function, that captures many natural settings. We derived a graphical model for this problem setting and formalized it as an instance of a POMDP. DynaMITE-RL is designed to exploit the causal structure of this model, and in a didactic GridWorld environment and several challenging continuous control tasks, we demonstrated that it outperforms existing meta-RL methods w.r.t. both learning efficiency and test-time adaptation in both online and offline-RL settings.

There are a number of exciting directions for future research building on the DLCMDP model. While we only consider Markovian latent dynamics in this work (i.e. future latent states are independent of prior latent states given the current latent state), we plan to investigate richer non-Markovian latent dynamics. We are also interested in exploring hierarchical latent contexts in which contexts change at different timescales. Finally, we hope to extend DynaMITE-RL to other real-world applications including recommender systems (RS), autonomous driving, multi-agent coordination, etc. DLCMDPs are a good model for RS as recommender agents often interact with users over long periods of time during which the user’s latent context changes irregularly, directly influencing their preferences.

CHAPTER 8

Chapter 8

Robometer: Scaling General-Purpose Robotic Reward Models via Trajectory Comparisons

8.1 Introduction

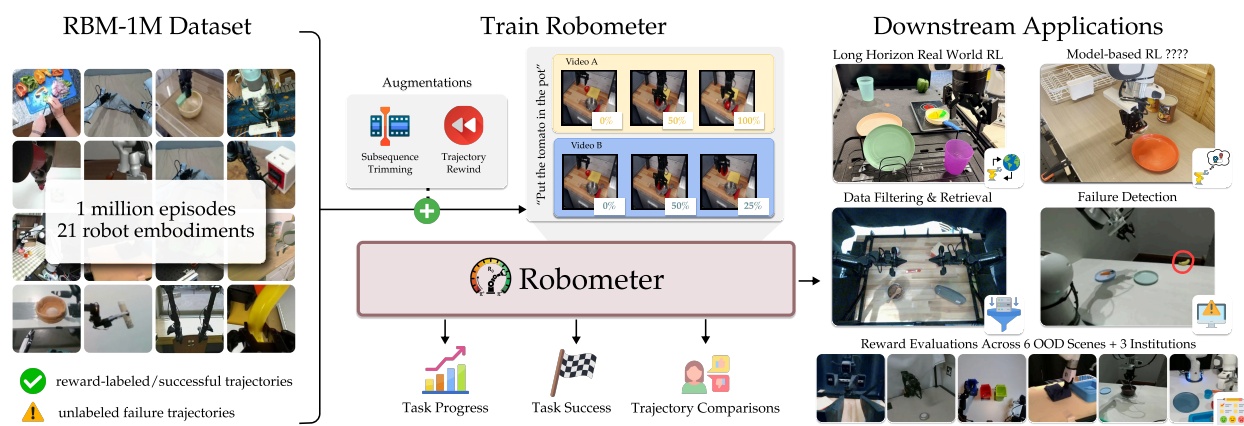


Figure 8.1: Robometer overview. Robometer is trained on RBM-1M, a 1M-trajectory dataset spanning 21 robot embodiments, containing both reward-labeled expert trajectories and reward-unlabeled failed trajectories. The model is supervised with a dual objective: predicting frame-level task progress (reward) and learning trajectory-level preferences from pairwise comparisons. It is also trained to predict per-frame task success for downstream RL.

In human cognition, comparative judgments are a core mechanism for internalizing calibrated

scales [169, 260, 270], enabling reasoning about relative progress and outcomes rather than isolated states. Analogously, the supervision signals used to train robotic reward models determine how well they internalize notions of task progress, enabling downstream applications such as on-line reinforcement learning (RL) [315, 333], imitation learning (IL) from noisy data [47, 212], automated failure detection [101], and offline RL [289]. Current general-purpose robot manipulation reward models rely exclusively on absolute progress labels derived from expert or reward-labeled demonstrations, providing pointwise, *trajectory-local* supervision [175, 274, 329, 333].

Such labels are easy to obtain for expert trajectories—for example, by linearly interpolating progress from 0 to 1—but become ill-defined and costly to annotate for failed attempts, where progress may fluctuate over time. As a result, large amounts of suboptimal data—ubiquitous in real-world robot learning—cannot be effectively leveraged [282]. This reliance on *trajectory-local* progress supervision limits both scalability and generalization. In this work, we address this limitation by training reward models with an additional *global* supervision signal that improves generalization across embodiments, scenes, and varying trajectory quality.

Our key insight is that *preference prediction* over trajectory pairs provides complementary supervision. While progress labels anchor reward values along individual trajectories, pairwise comparisons impose ordering constraints across diverse trajectories, tasks, robots, and viewpoints. This formulation enables learning from previously unusable suboptimal data by requiring only relative comparisons—curated without additional human annotation—rather than absolute scores. Specifically, trajectory comparison supervision (1) enforces consistent ordering across trajectories, providing global grounding beyond individual rollouts, and (2) scales naturally to unlabeled failed trajectories where absolute progress is ambiguous, resulting in better-calibrated rewards.

To instantiate this insight, we propose a scalable training recipe for reward modeling based on a dual reward-prediction objective: a frame-level progress loss on expert demonstrations and a preference-prediction loss over trajectory comparisons. Using this recipe, we train Robometer, a manipulation-centric reward model. Our training recipe reveals a strong mutual reinforcement

effect between the two supervision signals. Even when trained only on expert demonstrations, preference supervision improves Robometer’s ability to distinguish successful from suboptimal trajectories, suggesting that global comparative constraints induce a more structured internal reward representation. Moreover, as additional unlabeled suboptimal data is introduced, Robometer naturally scales to further improve performance.

To train Robometer, we curate RBM-1M, a large-scale reward-learning dataset for robot manipulation containing over one million trajectories collected from 21 robot platforms, including single-arm, bimanual, and mobile manipulators, as well as human demonstrations (see Figure 8.1). Importantly, RBM-1M intentionally includes a substantial number of suboptimal and failed trajectories that naturally arise during real-world data collection but are difficult to leverage with purely absolute progress-based supervision. The scale and diversity of RBM-1M are therefore critical for learning globally consistent preference relations across embodiments, tasks, and viewpoints while fully exploiting failure data that would otherwise be discarded. Beyond real-world failures, we further construct preference pairs using a suite of augmentations—including video rewinding [333] and cross-task comparisons—that expose the model to diverse successful and suboptimal behaviors. Across external benchmarks and our own evaluation trajectories collected from six out-of-distribution scenes spanning three institutions, Robometer outperforms state-of-the-art baselines by an average of 14% in reward rank correlation and achieves a 32% relative improvement in distinguishing successful from suboptimal trajectories.

Finally, we show that Robometer outperforms relevant baselines in real-world robot learning applications across a diverse set of downstream applications: (1) automatic online RL, (2) offline RL with noisy and expert trajectories, (3) dataset filtering for imitation learning, and (4) zero-shot failure detection across multiple robot embodiments and institutions. Overall, policy learning experiments with Robometer demonstrate $2.4 - 4.5\times$ higher success rates than the best baseline in each category. We publicly release Robometer, RBM-1M, and the training code at <https://robometer.github.io>, enabling the community to train their own models using our recipe or to directly use Robometer for robot manipulation applications.

8.2 Related Works

Learning reward functions is a central problem in reinforcement learning and robotics, and prior work has explored a wide range of approaches that differ in both the form of supervision and the scope of generalization.

Reward Learning from Demonstrations. Learning reward functions from human supervision is a long-studied topic in inverse RL (IRL), where reward functions are inferred from human demonstrations [1, 34, 86, 227, 348] or from expert and goal-state distributions [89, 90, 92, 118, 137, 273]. However, most IRL methods require task-specific expert demonstrations, necessitating the collection of new demonstrations whenever the task or reward specification changes. In contrast, we propose a framework for training reward models that can generalize effectively to new tasks.

Preference-Based Reward Learning. Prior work in psychology has observed that humans often use relative comparisons over absolute numerical scales when making judgments [169, 260, 270]. This insight has been central to modern reinforcement learning from human feedback (RLHF), where preference supervision from humans is used to learn reward models that are only identifiable up to monotone transformations, yet sufficient for effective policy optimization [16, 31, 56, 115, 159, 174, 223, 253, 317]. In contrast to these works, which treat preferences as the primary supervision signal to train domain-specific reward models, we generate preference supervision from both synthetic and real trajectories for which assigning progress labels is difficult. We use this preference signal as an *auxiliary* objective that complements direct progress prediction, enabling Robometer to learn from large, heterogeneous datasets without additional human preference supervision. Closest in spirit, Kwok et al. [166] generate synthetic preference labels from action mean-squared-error to train a robot action verifier, and Venkataraman et al. [289], Wang et al. [298] use vision-language models (VLMs) to generate frame-level preferences as the primary supervision signal for task-specific reward models. In contrast, we generate preference supervision by comparing entire trajectories and use it as an auxiliary signal to learn a

general-purpose reward function across tasks and embodiments.

Rewards from Foundation Models. Finally, recent work has sought to construct more general reward functions that operate directly on images, videos, and language. Large language models (LLMs) and VLMs have been applied to reward design, for example by generating executable reward code or shaping functions from natural language descriptions [211, 307, 326], directly producing reward functions [168], or guiding reward computation via language-conditioned state masking [129]. However, most of these approaches assume access to privileged state information, which is often unavailable or difficult to obtain in real-world robotic deployment settings.

To overcome this challenge, some approaches use task progress as a proxy reward, either by applying pre-trained VLMs as zero-shot progress or success estimators [3, 48, 64, 77, 93, 102, 210, 212, 213, 248, 266], or by training domain-specific models with progress-prediction objectives [47, 64, 82, 127, 152, 209, 225, 229, 315, 333]. However, directly using pre-trained VLMs for zero-shot video-language reward prediction often yields noisy or inconsistent signals [44, 175, 333], while smaller per-task models tend to overfit to domain-specific visual and semantic features, limiting generalization.

Recent methods address these limitations by fine-tuning VLMs or VLAs, enabling progress prediction to leverage visual-semantic representations learned from diverse data [97, 133, 175, 274, 329, 335] and demonstrating improved sample efficiency for RL. VLAC [329], for example, co-trains a VLA with relative progress difference targets, while $\pi_{0.6}^*$ [133] and Ghasemipour et al. [97] train distance-to-goal value functions that encode task progress. RoboReward [175] fine-tunes a VLM to predict discretized (1-5) progress labels generated via counterfactual instruction labeling by closed-source VLMs, and RoboDopamine [274] similarly fine-tunes a VLM for progress prediction but requires a goal image in addition to the language instruction and task-specific one-shot fine-tuning. In contrast, we introduce a more scalable training framework which uses an explicit auxiliary *preference prediction* objective, enabling us to scale to datasets containing suboptimal or failed trajectories without reward labels. Using such failed trajectories, even as just in-context learning examples for pre-trained VLMs, has been demonstrated to help reward models align

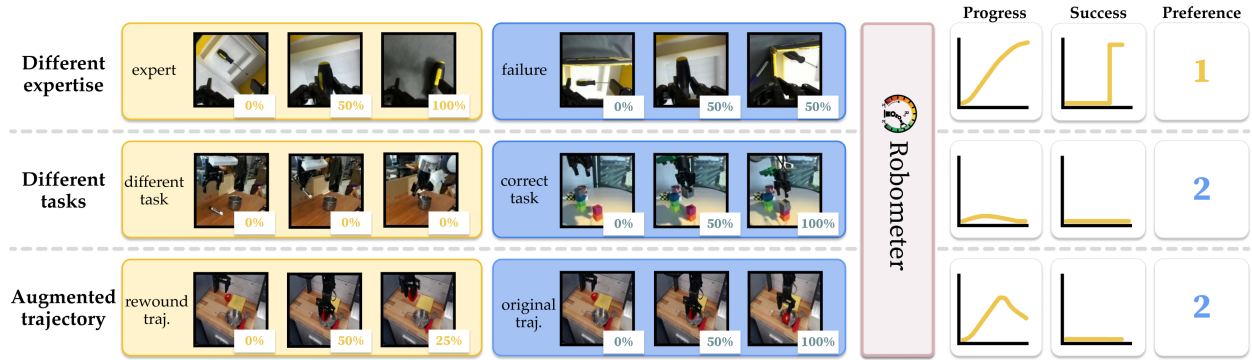


Figure 8.2: Robometer is a VLM-based reward model that predicts dense, per-frame progress-based rewards and success labels for the first of two video trajectories. To scalably train with failure data, we also predict which of the two video trajectories better completes the task (preference). We use three strategies for curating training examples from our given datasets, which are further detailed in Section 8.3.4 with model architecture shown in Appendix Figure F.2. At inference, the model can input up to two trajectories: with a single trajectory, it outputs per-frame progress & success predictions; with two, it additionally predicts a preference for which trajectory best performs the task.

more closely with human-specified rewards [282].

8.3 Robometer

We propose a *scalable recipe* for training reward models, along with **Robometer**, an instantiation of this recipe that provides dense rewards for robot manipulation. Our approach rests on three pillars: a diverse dataset which includes unlabeled failure trajectories, a pre-trained VLM backbone for generalization, and a training objective that combines **per-frame rewards** with **trajectory preferences**.

8.3.1 RBM-1M Dataset

Notation. We define the dataset $\mathcal{D} = \{\tau_i\}$ of trajectories, where each $\tau = \{o_{1:T}, l, p\}$ contains image observations o , a language instruction l , and a scalar progress label $p \in \{\text{None}, [0, 1]\}$ corresponding to the progress at the end of the trajectory. For expert demos, $p = 1.0$; for datasets with partial progress labels (e.g., RoboArena [10]), we use the provided score. For unlabeled failed trajectories, we set $p = \text{None}$.

Data Composition. Rather than maximizing trajectory quantity, RBM-1M focuses on view-

point, scene, and embodiment diversity. We aggregate 1 million trajectories from: (1) **Expert robot data** from diverse, multi-robot sources such as Open-X [61] and subsets of high-quality, single-robot datasets such as AGIBotWorld [62]; (2) **Human videos** from datasets such as Epic-Kitchens [67] for scene diversity or human-robot paired datasets like RH20T [83] to promote embodiment-invariant representations; (3) **Simulation** data from sources like LIBERO [197]; and (4) **Failed trajectories** from automated policy rollouts [343] and failure-detection datasets [195].

Our dataset overall includes 21 robot embodiments and over 1 million trajectories, hence RBM-1M. We also construct two evaluation datasets, RBM-EVAL-ID and RBM-EVAL-OOD, detailed in Section F.1.3. For further details on dataset filtering and aggregation, see Section F.1. We also list all dataset categories and trajectory counts in Appendix Figure F.1, Table F.4.

8.3.2 Robometer Architecture and Tokenization

Robometer builds on a causally masked VLM, QWEN3-VL-4B-INSTRUCT, to process either one video (for reward inference) or a pair of videos (for preference training).

Hidden Embedding Extraction. To extract rewards without disrupting the VLM’s pre-trained internal representations, we insert new, learned tokens into the sequence. We interleave **progress tokens** ($\langle | \text{prog_token} | \rangle$) within the first video sequence and a single **preference token** ($\langle | \text{pref_token} | \rangle$) at the end of the multi-video prompt:

$$\text{Tok}(l, o^1, o^2) \rightarrow \text{Tok}(l) \langle | \text{video_start} | \rangle \left[\text{Tok}(o_t^1) \langle | \text{prog_token} | \rangle \right]_{t=1}^T \langle | \text{split_token} | \rangle \left[\text{Tok}(o_t^2) \right]_{t=1}^T \langle | \text{pref_token} | \rangle, \quad (8.3.1)$$

where $\langle | \text{video_start} | \rangle$ is the model’s default image-start delimiter and $\langle | \text{split_token} | \rangle$ is a separator. The causal mask ensures that $\langle | \text{prog_token} | \rangle$ tokens attend only to current/previous frames of o^1 , producing dense, frame-level progress estimates for online reward inference, while $\langle | \text{pref_token} | \rangle$ attends to both trajectories to make a relative judgment. We fix both trajectories to length $T = 8$ to avoid preference predictions that rely on trajectory length as a proxy for quality. Progress tokens are inserted only for o^1 since at inference time, progress is predicted for a single trajectory; furthermore, if we insert progress tokens between o^2 frames, they would attend to o^1 .

8.3.3 Training Objectives

We optimize Robometer using a composite loss: $\mathcal{L} = \mathcal{L}_{\text{pref}} + \mathcal{L}_{\text{prog}} + \mathcal{L}_{\text{succ}}$. This allows the model to anchor rewards to absolute progress while learning to distinguish subtle quality differences through trajectory comparisons across the dataset.

Preference Prediction. We train a binary classifier, MLP_{pref} , on the hidden state $h_{\langle|\text{pref_token}|\rangle}$ of the $\langle|\text{pref_token}|\rangle$ to predict which trajectory better satisfies l :

$$\begin{aligned} \mathcal{L}_{\text{pref}} = - & \left[\mathbb{I}_{y=1} \log \sigma(\text{MLP}_{\text{pref}}(h_{\langle|\text{pref_token}|\rangle})) \right. \\ & \left. + \mathbb{I}_{y=2} \log(1 - \sigma(\text{MLP}_{\text{pref}}(h_{\langle|\text{pref_token}|\rangle}))) \right], \end{aligned} \quad (8.3.2)$$

where y is the ground-truth preferred trajectory.

Progress and Success. For the first trajectory o^1 , we attach an MLP head to each $h_{\langle|\text{prog_token}|\rangle,t}$ to predict continuous progress p_t and binary success s_t . Similar to prior work [47, 329, 333], we define per-frame progress targets $p_{1:T}$ for expert demonstration data, where the final target progress $p = 1$. To better model multi-modal reward/progress distributions than continuous progress prediction, we discretize progress into $N = 10$ uniformly spaced bins over $[0, 1]$ and model progress prediction as a categorical distribution [23, 84, 133]. For a trajectory of length T , the ground-truth continuous progress target at frame t is defined as $p_t = t/T$ for $t \in \{1, \dots, T\}$. This scalar target is projected onto a categorical distribution over N bins using linear interpolation between the 2 neighboring bin centers. The progress head $\text{MLP}_{\text{progress}}$ outputs a categorical distribution $\hat{p}_t \in \Delta^N$, and the progress loss is computed using cross-entropy:

$$\mathcal{L}_{\text{prog}} = \frac{1}{T} \sum_{t=1}^T \text{CE}(\text{Project}(p_t), \text{MLP}_{\text{progress}}(h_{\langle|\text{prog_token}|\rangle,t})).$$

At inference time, a continuous progress estimate is recovered by taking the expectation over the bin centers, $\hat{p}_t = \sum_{i=1}^N z_i \hat{p}_{t,i}$, where $\{z_i\}_{i=1}^N$ denote the fixed bin centers. Per-frame success targets are defined such that $s_t = 0$ for $t < T$ and $s_t = 1$ for $t = T$. In some datasets, the human operator stops recording a trajectory several frames after the task has already been completed; to improve the accuracy of both progress and success prediction, we sample 10 trajectories per data source to determine the point at which the task actually ends (Appendix Section F.1.1). We

train success prediction with binary cross-entropy on s , with balanced class weights adjusted per-batch to account for negative sample imbalance:

$$\mathcal{L}_{\text{succ}} = \text{BalancedBCE}(s_{1:T}, [\text{MLP}_{\text{success}}(h_{\langle \text{prog_token} \rangle, t})]_{1:T}).$$

8.3.4 Data Sampling and Augmentation

Given these losses, the ideal training regime for Robometer would rely on large-scale, preference-labeled robot trajectory datasets containing explicit progress-labeled failures. Such failures are particularly important because, at deployment time, reward models are frequently queried on out-of-distribution trajectories—e.g., failures induced by online RL exploration, compounding execution errors, or noisy data collection—that deviate substantially from the training distribution. In practice, however, preference annotations over robot trajectories are limited, and dense per-frame progress labels for failed executions are expensive and difficult to obtain. We address this limitation by constructing training inputs (l, o^1, o^2) and targets y dynamically from RBM-1M using three complementary strategies displayed in Figure 8.2:

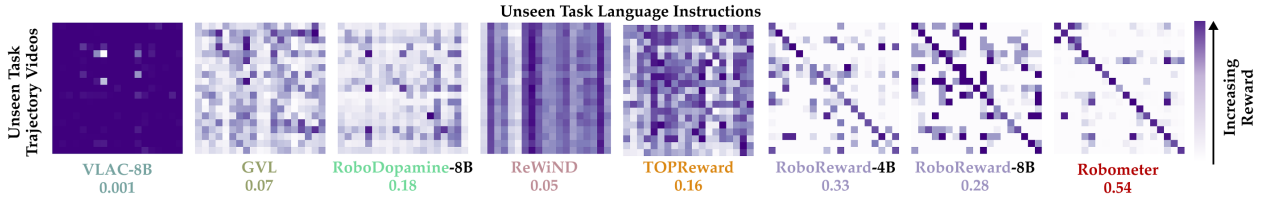


Figure 8.3: Video-Language Reward Confusion Matrix. For each task sampled at random from *self-collected*, *unseen* data from RBM-EVAL-ODD, we compute rewards for all combinations of demonstration videos and language descriptions. **Robometer** produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions. We also report the column-normalized diagonal mean under each model, which represents the fraction of the model’s total reward for aligned task and video pairs.

1. **Progress-Based Comparisons (Different Expertise).** To teach the model to distinguish execution quality, we sample two trajectories τ_1, τ_2 sharing an instruction l but differing in outcome (e.g., an expert demonstration $p=1$ vs. an unlabeled failure $p=\text{None}$) or progress ($p^1 \neq p^2$). We set the preference target $y=1$ if $p^1 > p^2$ (or if τ_1 is the expert), and $y = 2$ otherwise. This allows Robometer to leverage unlabeled failures by contrasting them against

successful demonstrations.

2. **Instruction Negatives (Different Tasks).** To ensure rewards are grounded in the language command, we sample τ_1 and τ_2 with distinct instructions $l^1 \neq l^2$. We randomly select one instruction as the conditioning text l , set the preference label y to the trajectory corresponding to the selected instruction, and set the progress target $p_t=0$ for the other, enforcing that correct behavior for the wrong task yields no reward.
3. **Video Rewind (Augmented Failures).** To explicitly model “undoing” progress—a common failure mode in RL—we generate synthetic preferences from a single expert trajectory τ by reversing a segment of time. Prior work denotes this type of augmentation as *video rewind* [47, 124, 333]. We sample indices $1 \leq t_1 < t_2 < t_3 \leq T$ to form a *Chosen* forward sequence $o^c = o_{t_1:t_3}$ and a *Rejected* rewind sequence, either $o^r = [o_{t_1:t_3}, o_{t_3-1:t_2}]$ or $[o_{t_3:t_1}]$. The Chosen and Rejected sequences are randomly assigned to o^1 and o^2 to construct the preference label. While o^c targets linear forward progress, we explicitly penalize the reversal in o^r by assigning decreasing progress targets matched to their frame indices.

Fixed-length Subsampling. To avoid biasing toward longer trajectories, we construct fixed-length inputs by randomly selecting start and end indices in the trajectory, and uniformly sampling T frames between them.¹

Summary. Robometer builds upon a base VLM, QWEN3-VL-4B-INSTRUCT, and inserts additional learnable tokens to predict preference, progress, and success. We train Robometer on RBM-1M, a dataset consisting of both progress-labeled and progress-unlabeled data, by sampling trajectory comparisons across the entire dataset. These comparisons come from failure trajectories, comparisons across different tasks, and rewind videos. At inference, Robometer can operate on either one or two trajectories: with a single trajectory, it outputs per-frame progress and success predictions; with two trajectories, it additionally predicts a preference while still producing per-frame predictions for the first trajectory. For additional model implementation and training details, see [Section F.2](#).

¹Robometer is still a *dense*, per-frame reward model: during inference, we query over $o_{1:t} \forall t \in [1, \dots, T]$ for a trajectory of length T to produce T rewards. Our subsampling always includes the first and current frame.

		w/ Varied Training Data				w/ RoboReward Training Data			w/ our RBM-1M data	
		GVL	VLAC	TOPReward	RoboDopamine-8B	RoboReward-4B	RoboReward-8B	Robometer	ReWiND	Robometer
(a) VOC $r \uparrow$	RBM-EVAL-ID	0.16	0.16	0.42	0.79	0.77	0.82	0.84	0.46	0.92
	RBM-EVAL-ODD	0.21	0.17	0.40	0.80	0.88	0.88	0.93	0.51	0.94
(b) Kendall $\tau_a \uparrow$	RBM-EVAL-ODD	0.19	0.08	0.13	0.45	0.50	0.47	0.55	0.01	0.64

Table 8.1: (a) Reward alignment (VOC Pearson r) and (b) trajectory ranking (Kendall τ_a) on RBM-EVAL datasets. Baselines are split into categories based on training data: **GVL** and **TOPReward** training data is unknown, **VLAC** is trained on a 300k-trajectory dataset, and **RoboDopamine** is trained on a 100k-trajectory dataset. We compare **Robometer** against **RoboReward-4B/8B** with their own training data, and we also evaluate **ReWiND** and **Robometer** trained with the full RBM-1M dataset. Kendall τ_a is not calculated for RBM-EVAL-ID due to it only having simulation failure data.

8.4 Experiments

Our experiments aim to study Robometer’s effectiveness in producing rewards for robot learning. Specifically, we organize our experiments to answer the following questions:

- (Q1) **Reward Evaluation:** How well do Robometer rewards reflect task progress on *unseen* tasks and embodiments?
- (Q2) **Ablation + Analysis:** How much does each component of Robometer contribute to reward performance?
- (Q3) **Policy Learning:** How does Robometer compare against baselines in enabling downstream robot learning?

Baselines. We compare **Robometer** against the strongest set of video-language input, zero-shot-capable, and open-sourced or API-accessible reward baselines described in Section 8.2. We list a dataset size comparison table for baselines and related papers in Appendix Table F.1.

- **VLAC-8B** [329] trains a VLA that predicts actions and rewards on a dataset of 300k human and robot trajectories. We compare against their larger 8B parameter checkpoint.
- **GVL** [212] prompts a pre-trained closed-source LLM with shuffled video frames to predict task progress for subsampled frames across the video sequence. We use GPT-5 mini as it is the best-performing closed-source model on the RoboRewardBench reward evaluation benchmark [175].

- **TOPReward** [48] prompts a pre-trained VLM (Qwen-3-VL 8B) with “does the trajectory complete the task?” and uses the log-likelihood of the “true” token as the reward.
- **ReWiND** [333] trains a small transformer-based network with a direct progress prediction objective along with video rewinding to simulate failed policy rollouts. We train ReWiND with RBM-1M to maximize its zero-shot capability.
- **RoboDopamine-8B** [274] fine-tunes a VLM for reward prediction via “frame hops” comparing forward and rewind frames. While designed for reward prediction conditioned on a *goal image* and instruction, we evaluate it in our zero-shot setting without a goal image for fair comparison. We use the latest, 8B parameter, RoboDopamine-2.0 checkpoint.
- **RoboReward-4B/8B** [175]: Fine-tunes a Qwen-3-VL 4B/8B VLM for discrete (1-5) progress prediction on a dataset consisting of data from OXE [61] and RoboArena evaluations [10]. Generates *counterfactual* instructions via closed-source VLMs to simulate failed trajectories.

Custom Evaluation Datasets. We train Robometer, and certain baselines when applicable, on the aforementioned RBM-1M dataset. Prior large-scale reward modeling baselines mainly evaluate on validation or test set versions of the datasets they train on [175, 274, 329, 333], which contain in-distribution arms, camera angles, or scenes. Instead, we collect our own evaluation dataset, RBM-EVAL-OOD, consisting of 976 trajectories collected from 3 academic institutions that are guaranteed not to be in the training data of *any* baseline, consisting of 6 embodiments (including human hands), 3 of which are not in RBM-1M, collected across diverse camera angles. We also aggregate an in-distribution test split of unseen trajectories collected from datasets in RBM-1M, denoted RBM-EVAL-ID. See more evaluation dataset details in [Section F.1.3](#).

8.4.1 Reward Evaluation

As detailed in [Section 8.3.2](#), our focus is on training a reward model which: (1) generalizes to new tasks, embodiments, and domains while (2) providing reward feedback useful for *policy learning*. We structure this subsection to highlight Robometer’s strong performance across both criteria.

Trajectory Task Alignment. Our first main result demonstrates that Robometer accurately

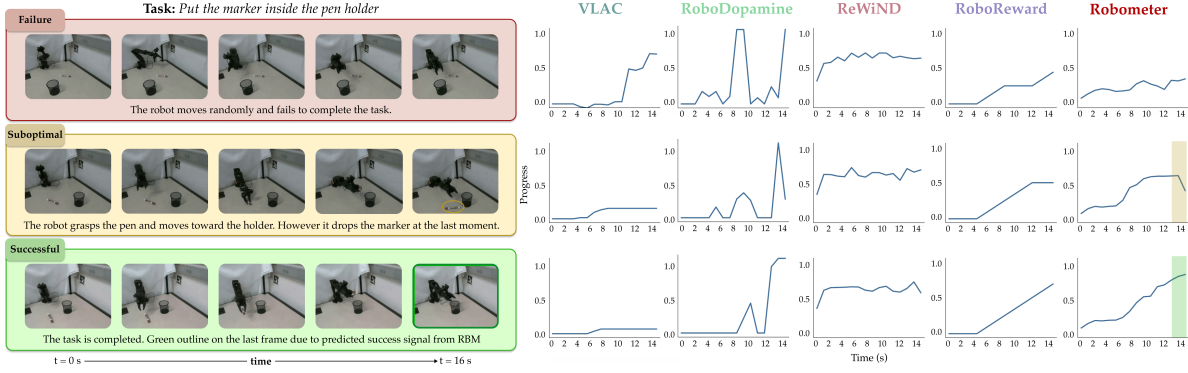


Figure 8.4: Qualitative Analysis of Failure, Suboptimal and Successful Trajectories. We visualize the progress predictions for three trajectories of different quality for the same task. Notably, for the suboptimal trajectory, **Robometer** predicts steadily increasing progress as the robot approaches the pen holder, but sharply reduces its progress estimate when the marker is dropped, correctly reflecting regression in task completion. In contrast, **RoboReward** continues to assign high progress despite the task failure. Finally, **Robometer** is the only model that correctly predicts task success for the successful trajectory (i.e., high final progress value and explicit success prediction).

distinguishes between different tasks in RBM-EVAL-ODD, which directly reflects its ability to assign rewards that align with task semantics, even across unseen robot embodiments, camera viewpoints, and scenes. We plot confusion matrices comparing unseen, successful trajectory videos versus their language instructions in Figure 8.3. Ideally, a **purple diagonal** indicates correct video-instruction pairs, with low (white) values elsewhere. Robometer clearly produces the strongest disparity between the diagonal and off-diagonal elements, highlighting its superior ability to reward a robot for performing the *correct* task, which is especially important in cluttered, multi-task settings. This ability is in part due to how we sample *different-task* negative preference and progress examples across the entire dataset (cf. Section 8.3.4).

Reward Alignment. Quantitatively, we evaluate the ability of baselines to predict increasing progress for *successful* robot videos from both RBM-EVAL-ODD and RBM-EVAL-ID in Table 8.1(a). We report Value Order Correlation (VOC) [212] $\in [-1, 1]$, which calculates the Pearson correlation of predicted rewards for each trajectory video frame against their ground-truth timestep value. Overall, Robometer performs the best across the board on both test sets, especially on RBM-EVAL-ODD. We break down per-dataset and per-subset results for both test sets in Section F.3, and evaluate on an external RoboRewardBench [175] benchmark in Section F.3.3.

Relative Trajectory Rankings for Mixed Expertise Data. Next, we quantitatively demonstrate that Robometer is more effective than baselines at providing rewards useful for *policy learning*. For a robot policy to learn with rewards, the rewards should not only be high when performing the correct task, but also be low for incorrect execution. We measure this using the Kendall- τ_a coefficient [222], an ordering metric $\in [-1, 1]$ robust to ties. We calculate the alignment between model-assigned final rewards and the ground-truth ordering between failed, suboptimal, and successful trajectories for the same task. A higher τ_a value demonstrates that the reward model more accurately distinguishes between levels of policy performance and thus can provide proper reward signals to the policy for both low- and high-quality behaviors.

We report results in Table 8.1(b). On RBM-EVAL-00D, Robometer achieves a Kendall- τ_a of 0.66, substantially outperforming RoboReward-4B (0.50) and RoboReward-8B (0.47), indicating that Robometer more reliably recovers the correct ordering among failed, suboptimal, and successful trajectories. Notably, even when trained on the same data as RoboReward, Robometer outperforms RoboReward in both policy ranking and reward alignment, highlighting the effectiveness of our data augmentation strategies.

To further illustrate this behavior, we visualize reward predictions over time for failed, suboptimal, and successful trajectories in Figure 8.4. Robometer exhibits sharper separation in rewards between different levels of execution and accurately reflects regression in task progress. Additional results evaluating preference prediction accuracy are reported in Section F.3.

Reward Fine-tuning. Finally, we demonstrate that Robometer serves as a good initialization for domain-specific fine-tuning. We fine-tune on RoboFAC [203], a dataset of robotic failures and corrections spanning 16 tasks and 53 scenes (11k trajectories), including both simulated and real-world successes and failures. We adapt Robometer via LoRA [122] adapters and full fine-tuning (FFT). We compare against fine-tuning the base VLM Qwen/Qwen3-VL-4B-Instruct in the same way.

In addition to the previous VOC r and Ranking Kendall τ_a , we also compare a *success - fail* metric measuring the difference in final reward between successful and failed trajectories of the

same task. Qwen3 fine-tuned still performs worse than Robometer zero-shot on 2 out of 3 metrics; fine-tuning from Robometer yields substantially better reward evaluation results than training Qwen3-VL from scratch across all of our ranking metrics (Table 8.2). Importantly, LoRA and FFT perform similarly, demonstrating that Robometer can be effectively fine-tuned with just 1 GPU. See further experiment details in Section F.6.

Method	VOC $r \uparrow$	Kendall $\tau \uparrow$	Succ-Fail Diff $\uparrow$
Robometer-4B (Zero-shot)	0.652	0.436	0.141
Qwen3-VL (LoRA)	0.701	0.067	0.005
Qwen3-VL (Full FT)	0.727	0.102	0.008
Robometer-4B (LoRA)	0.875	0.786	0.271
Robometer-4B (FFT)	0.884	0.802	0.302

Table 8.2: Finetuning Robometer on RoboFAC dataset. Zero-shot performance is strong, and fine-tuning the base Qwen3 VLM performs worse on Kendall τ and success - fail difference compared to zero-shot **Robometer**. Meanwhile, **Robometer** fine-tuned performs best, either with LoRA or FFT.

Overall, our results demonstrate that **Robometer outperforms baselines** in both **generalization** and **distinguishing** successful / failed trajectories, and that it also serves as a **strong initialization for fine-tuning**. We next analyze *why*.

8.4.2 Ablations: Why does Robometer Perform so Well?

Here, we investigate individual components of Robometer to evaluate specific hypotheses about reward model training and its effects on downstream RL performance.

- H1** Predicting **preferences** (Equation (8.3.2)), even without paired failure trajectories, improves reward performance.
- H2** Scaling preference prediction with additional **failure data** leads to improved reward model performance.
- H3** Fine-tuning from **pre-trained VLMs** helps with reward predictions on unseen tasks.

Our main analysis is performed via a controlled setting with data from the LIBERO [197] robot manipulation simulated benchmark. We train models with 1,709 successful demos from

LIBERO-10, Object, Goal, Spatial} and evaluate performance on a sample of the 8,262 unseen, paired, successful and failed trajectories from LIBERO-90.

Ablation	(a) LIBERO-90			(b) RBM-EVAL-00D		
	VOC r	Kendall τ	Suc – Fail	VOC r	Kendall τ	Suc – Fail
H1 Prog. Only	0.96	0.63	0.11	0.93	0.31	0.08
H1 +Preference	0.90	0.74	0.22	0.95	0.54	0.24
H2 +Failed Data	0.98	0.92	0.46	0.94	0.64	0.32
H3 ReWiND Arch.	0.48	-0.14	-0.02	0.51	0.01	0.02

Table 8.3: Reward alignment (VOC Pearson r), policy ranking (Kendall τ), and average reward difference between successful and failed trajectories on LIBERO-90 and RBM-EVAL-00D.

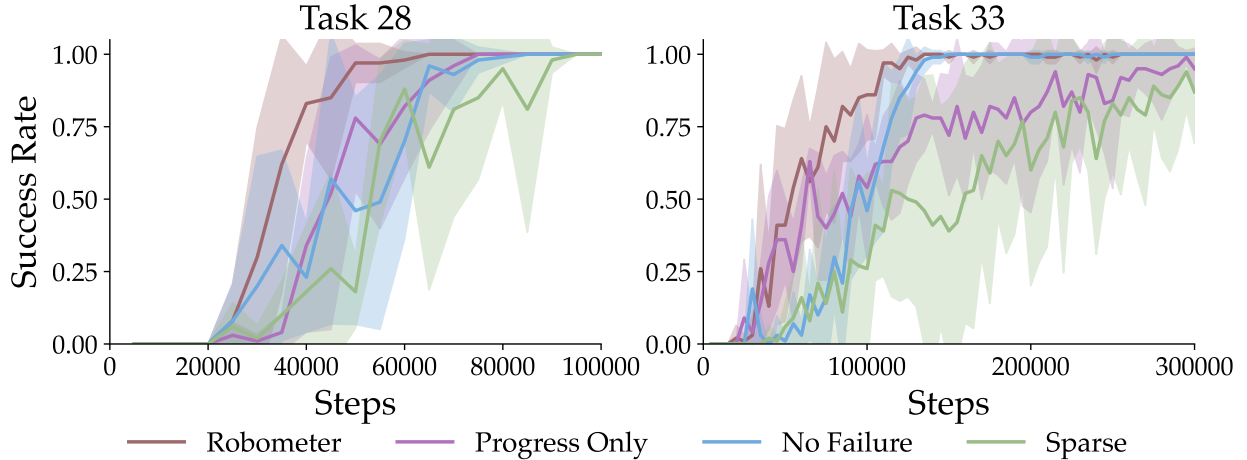


Figure 8.5: RL w/ Ablation Models in LIBERO-90 tasks from scratch, corresponding to ablations trained only on LIBERO-10/Object/Goal/Spatial data from Table 8.3. We report the average success rate $\pm$ standard deviation across 5 seeds.

Reward Model Ablations. To measure **H1**, we train Robometer with *only* progress prediction and also Robometer with both progress and preference prediction on the 1,709 demo dataset containing no failed trajectories. We then measure **H2**—scaling with failure data for preference training—by adding in 1,929 generated, failed LIBERO trajectories and train Robometer with the full Robometer training objective of progress and preference prediction on the larger dataset. These LIBERO ablations are trained with LoRA [122] due to the small dataset size. Finally, we

verify the importance of a pre-trained VLM (**H3**) by training a larger, 500M-parameter version of ReWiND’s transformer model (originally designed for low-data regimes) with our preference and progress objectives on the paired-failure LIBERO dataset.

We depict results on LIBERO, and separately, trained on the full RBM-1M and evaluated on RBM-EVAL-00D (with failed data removed for +Preference and +Failed Data), in [Table 8.3](#). First, comparing **H1 Prog. Only** to **H1 +Preference**, adding preference supervision consistently improves policy ranking performance, increasing Kendall- τ_a on both LIBERO-90 and RBM-EVAL-00D. Second, **incorporating failed trajectories** for preference training (**H2 +Failed Data**) yields the **largest gains** across all ranking-based metrics. On LIBERO-90, Kendall- τ improves to 0.92 and the average difference in final rewards between success-failure increases more than 4 $\times$ relative to progress-only training. Similar trends hold on RBM-EVAL-00D, where Kendall- τ and success-failure separation improve significantly. Finally, replacing the pretrained VLM backbone with a scaled variant of ReWiND’s architecture (**H3 ReWiND Arch.**) results in a severe degradation across all metrics, confirming that a large-scale multimodal pretrained backbone is essential for learning generalizable reward representations. See [Section F.4](#) for additional ablations.

Ablation Results on RL Performance. Before moving to our full policy learning experiments, we verify that the reward evaluation trends observed in our ablation studies hold for policy learning. We train policies via online RL using ablated reward models on two tasks from the unseen LIBERO-90 suite. These tasks were selected because sparse-reward RL eventually achieves a near-100% success rate, allowing us to directly compare *sample efficiency* rather than success rates.

Results in [Figure 8.5](#) demonstrate that improvements in reward evaluation metrics consistently transfer to policy success rates. For both tasks, policies trained using Robometer (the **H2** LIBERO model) demonstrate better sample efficiency than ablations (**H1** LIBERO models) and sparse reward, highlighting the importance of dense, well-calibrated reward signals for efficient policy optimization. Overall, Robometer trained only on LIBERO achieves 2 – 4 $\times$ **better sample efficiency** than sparse reward on these unseen tasks. Additional details of the RL training setup

are provided in Section F.5.

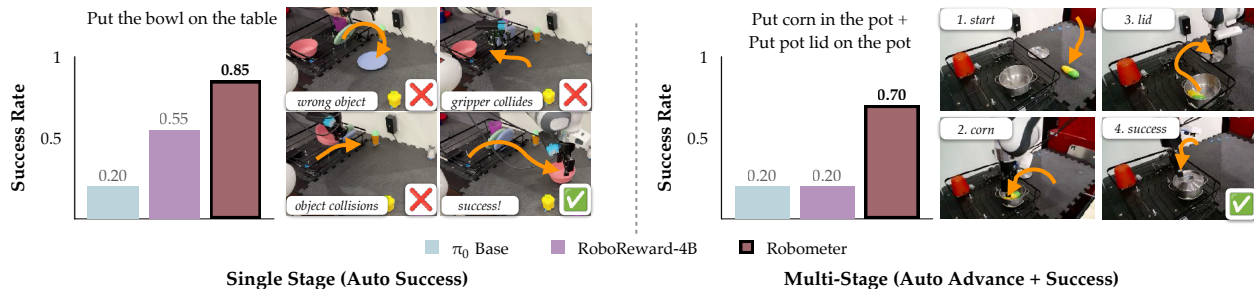


Figure 8.6: Automatic online RL with DSRL [291] on a DROID [149] setup with Robometer improves π_0 from 20% to 85% on a single-stage task and 20% to 70% on a two-stage task, outperforming RoboReward’s overall success rate by 2.5 $\times$. DSRL with Robometer learns to avoid base π_0 errors such as collisions or moving the wrong object. The setup is deemed “automatic” because success detection and stage advancement are handled automatically by the reward model, requiring human intervention only for physical scene re-sets.

8.4.3 Accelerating Robot Learning with Generalizable Rewards

We evaluate whether Robometer’s dense and generalizable reward signals can be used **zero-shot** into improved downstream robot learning across four settings, including both prehensile and non-prehensile tasks: (1) automatic online RL, (2) offline RL with mixed-expertise data, (3) data filtering and retrieval for policy improvement, and (4) out-of-distribution failure detection. Across all experiments, we compare against RoboReward-4B—the strongest baseline reward model in our offline evaluations—to assess how Robometer’s dense, instruction-aligned rewards affect learning stability, robustness, and sample efficiency. We also compare against strong, relevant, non-reward-model baselines for each setting where applicable. All policy learning results are averaged over 20 evaluation trials. Additional details and finer-grained results on each experiment can be found in Section F.5.

Automatic Online RL. First, we evaluate Robometer in an *automated* online RL setting by training DSRL [291] from scratch on a π_0 base policy [32] pre-trained on DROID [149]. Robometer enables autonomous RL by providing dense rewards and explicit *success predictions*, which we use to automate episode termination; manual human intervention is required only for physical scene resets. For comparison, RoboReward’s discrete scores are also used for both reward shaping and

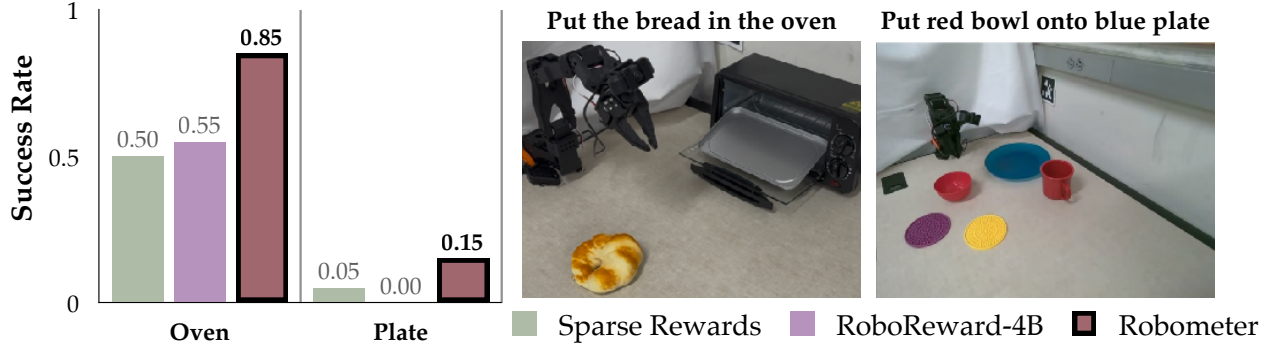


Figure 8.7: Offline RL results using IQL on a mixture of Noisy and Expert trajectories. **Robometer** rewards consistently outperform both **RoboReward** and sparse rewards: 2.4× average success rate improvement over the best baseline for each task.

success detection. As shown in Figure 8.6 (left), DSRL+Robometer improves success from 20% to 85% in ≤ 45 **minutes** (10k timesteps), outperforming RoboReward’s 55%. This gap arises from a key failure mode of RoboReward: it frequently assigns maximum rewards for unrelated tasks (e.g., picking up the wrong object), leading to premature resets and reinforcing incorrect behaviors. In contrast, Robometer provides a more reliable learning signal.

Next, we evaluate a **longer-horizon RL** setting in Figure 8.6 (right), where Robometer’s success predictions trigger progression between stages of a multi-stage task. Unlike methods that explicitly train with multi-stage rewards and thus require stage labels [47, 152], we simply decompose tasks into stages at inference time using a pre-trained VLM and use Robometer to advance stages automatically. In this setting, DSRL+Robometer improves π_0 ’s success from 20% to 70% over 10k timesteps, outperforming RoboReward’s 20%, which suffers from inaccurate rewards and unreliable stage transitions. Across both setups, Robometer outperforms RoboReward’s overall success rate by an average of 2.5×.

Finally, we perform an additional online RL experiment—*model-based RL* integrating Robometer into DreamZero [320]—where Robometer improves success rate from 20% to 70%. See details and results in Section F.7.

Combining Noisy and Expert Data via Offline RL. We consider an offline RL setting with mixed-expertise data for two tasks on an SO-101 robot (SO-101 is not in RBM-1M), combining expert and noisy, suboptimal demos, as shown in Figure 8.7. We train policies with Implicit Q-

Learning (IQL) [160] to study how dense rewards from Robometer improve learning stability and policy extraction in offline RL.

Accurate, dense reward signals can provide informative intermediate feedback, reducing reliance on long-horizon credit assignment and enabling trajectory “stitching” with smaller discount factors γ , thereby reducing value function variance. For each of sparse reward, RoboReward, and Robometer, we sweep $\gamma \in \{0.90, 0.95, 0.99\}$ and report the best-performing checkpoint over 30,000 offline training steps. We observe that Robometer, which provides dense, temporally aligned rewards, performs best at a lower discount factor $\gamma = 0.9$ and outperforms both baselines across both tasks with a $2.4\times$ success rate improvement over the best baseline in each. RoboReward performs similarly to sparse rewards across the γ sweep, as its categorical (1–5) outputs provide less dense guidance and often assign high rewards to suboptimal trajectories, leading to noisy training signals.

Data Filtering & Retrieval. We next evaluate Robometer as a mechanism for unsupervised data filtering and retrieval. Using a bimanual “play” dataset [206] of unannotated, multi-task trajectories collected on a Trossen AI setup (not in RBM-1M), we retrieve the top 100 subtrajectories for a given task instruction. We compare retrieval relevance against RoboReward, pre-trained SigLIP [330], and a retrieval-specific baseline, STRAP [215]. For Robometer we retrieve subtrajectories using (i) the preference objective via pairwise trajectory comparisons, or (ii) the progress objective by computing each trajectory’s value–order correlation. As shown in Figure 8.8(a), Robometer consistently achieves higher retrieval relevance across five tasks. Finally, we LoRA-finetune $\pi_{0.5}$ [122, 134, 202] on these retrieved segments. Policies trained on Robometer-filtered data vastly outperform those using baseline-retrieved data on Stir the Pot and Open the Red Drawer (Figure 8.8(b)), demonstrating its efficacy for targeted imitation learning. Low baseline success rates despite high retrieval rates stem from their retrieval of more failed and suboptimal, yet task-relevant, subtrajectories. Overall, Robometer-retrieval averages a $4.5\times$ higher success rate than the best baseline.

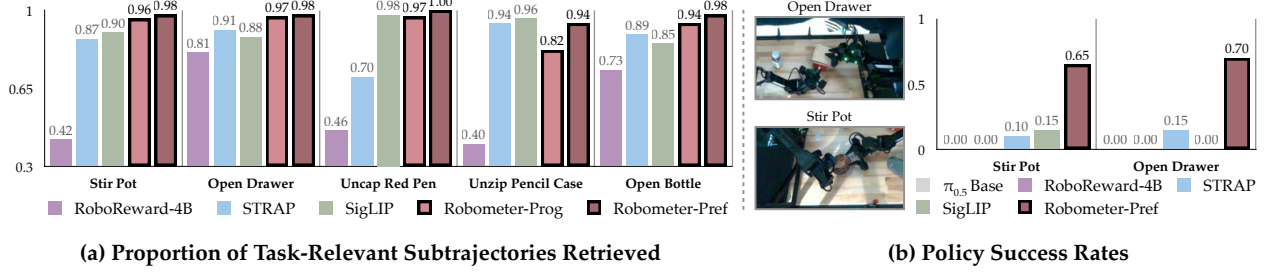


Figure 8.8: (a): Proportion of task-relevant subtrajectories out of 100 retrieval queries. Our method consistently retrieves a high number of relevant subtrajectories using either the preference or progress objective. **(b): Success rates** of LoRA-finetuned $\pi_{0.5}$ policies using the retrieved trajectories from each method. Small amounts of suboptimal & unrelated data retrieved by other baselines degrade policy-learning performance: **Robometer**-retrieval attains an average 4.5× success rate improvement over the best baseline.

Task	T.U.	VLAC	GPT-5-mini	RoboReward-4B	Robometer
move banana	0.53	0.45	0.48	0.91	0.94
move mouse	0.50	0.00	0.89	0.80	0.91
pour pebble	0.32	0.00	0.25	0.73	0.83
fold towel	0.58	0.16	0.27	0.40	0.58
pull tissue	0.43	0.00	0.00	0.57	0.76
put spoon	0.22	0.00	0.25	0.73	0.73
stir pot	0.47	0.00	0.17	0.95	0.90
Average	0.48	0.16	0.33	0.74	0.81

Table 8.4: Failure detection performance. **Robometer** achieves the highest average F1 score. T.U. is the token-uncertainty baseline.

Failure Detection. Detecting failures during online deployment is critical for safe deployment. Thus, we evaluate Robometer’s zero-shot failure detection on 100 manipulation trajectories from a Franka Panda DROID robot (30 successful, 70 failed) spanning 7 tasks collected in scenes not in RBM-1M. Failures are evenly split between irreversible failures (e.g., drops or spills) and insufficient-progress failures, where execution stalls or terminates prematurely. We compare our method against: the token-uncertainty [101] of π_0 -FAST-DROID [239] as proposed by Gu et al. [101] for zero-shot failure detection; VLAC, which reports failure detection results; GPT-5-mini; and RoboReward-4B. Failures are detected via temporal inconsistencies in predicted per-frame rewards.

As shown in Table 8.4, Robometer achieves the highest average F1 score, effectively balancing true positive and true negative rates (TPR and TNR); the full breakdown with TPR and TNR is provided in Appendix Table F.20. VLAC frequently flags trajectories as failures, achieving high TPR but low TNR, resulting in lower F1 scores. RoboReward-4B performs competitively but underperforms Robometer, particularly on tasks with subtle failure modes such as *fold towel* and *pull tissue*. As detailed and visualized in Section F.5.3, Robometer robustly detects irreversible, insufficient-progress, and non-terminal failures (e.g., hovering, oscillation, or partial completion), fully zero-shot across tasks and environments—unlike prior methods that require task-specific thresholds, calibration, or test-time interaction [6, 101, 309].

8.5 Limitations and Future Work

Robometer operates as a dense, per-frame reward model over subsampled video inputs, enabling scalable training but limiting its ability to memorize specific past key events for very long-horizon or partially observed tasks. In addition, real-world robot executions exhibit a wide diversity of failure modes, many of which are rare, subtle, or task-specific, and the current training data may not fully capture this breadth. As a result, Robometer may fail to recognize or correctly reward certain failure cases that fall outside the dominant patterns seen during training. We hope that our training recipe enables future work to build upon larger reward modeling datasets.

As a vision-language-based model, Robometer also lacks direct access to latent physical state such as contact forces, grasp stability, or compliance, and may fail to recognize or correctly reward failure cases driven by these factors until they become visually observable. Future work could address these limitations by incorporating denser temporal modeling, VQA-style supervision to reason about task structure and completion criteria, and off-domain data for better generalization [184, 332], as well as by developing more systematically curated failure datasets that better reflect the diversity of real-world failure modes [282]. Finally, real-world executions may also contain safety-critical states (e.g., in a constrained MDP formulation) that Robometer is not pre-trained to predict. Future work can train Robometer with an additional safety head, similar to

the success prediction head, on provided safety labels.

CHAPTER 9

Chapter 9

Concluding Remarks and Future Directions

A central goal for robot learning is to build systems that are both easy for people to guide and capable of improving from their own experience. Pretraining can provide a strong starting point, but it does not resolve the challenges that arise during deployment in real-world environments. In homes and workplaces, tasks are open-ended, user intent is often underspecified, and small variations in the environment can lead to failures. Addressing these challenges requires progress in both how robots interface with humans and how they learn over time.

Steerability. A key challenge is to design interfaces that allow non-expert users to guide robot behavior without requiring precise control or extensive demonstrations. While teleoperation and full demonstrations remain effective, they are difficult to scale in everyday settings. A more practical approach is to leverage signals that arise naturally during interaction.

Examples include trace-based guidance, where users specify desired motion directly in image space, and corrective interventions, where users step in during execution to redirect unsafe, undesired, or incomplete behavior. These interventions are especially important in real-world settings, where small corrections at critical moments can prevent failures without requiring a full new demonstration. More broadly, preference signals and multimodal interaction expand the

space of ways people can communicate intent without specifying every detail of execution.

Another important direction is to expose intermediate structure in the model’s decision-making. When task specifications are ambiguous, committing directly to actions can lead to unintended behavior. By surfacing intermediate plans or representations of intent, systems can allow users to inspect or revise decisions before execution. This creates a more transparent interface between human guidance and autonomous behavior, improving both controllability and safety.

Autonomous improvement. Steerability alone is not sufficient. Robots must also improve from experience after deployment, especially under distribution shift.

In realistic settings, failures, near-misses, and partial progress are inevitable. Rather than treating these as noise, they should be treated as signals for learning. This is particularly important for long-horizon tasks that combine locomotion and manipulation, where errors can compound and recovery behavior becomes essential.

Progress in this direction requires learning signals that remain meaningful across tasks and environments, as well as algorithms that can make use of diverse and imperfect data. Reward models that capture task progress, along with methods that learn from suboptimal trajectories, provide one path forward. More broadly, systems must be able to update behavior from interaction, rather than relying solely on pretraining or offline datasets.

Interaction between guidance and learning. The most promising direction lies in combining human guidance with autonomous improvement.

Instead of relying on static supervision, robots can use interaction as an ongoing source of learning. They can request feedback at critical moments, incorporate corrective interventions during execution, and use human input to guide exploration. For example, a robot might pause before executing an uncertain action, request clarification, or recover from a mistake using a brief user correction rather than a full demonstration.

This interaction creates opportunities for more efficient learning. Rather than requiring dense supervision, small corrections, preferences, or interventions can improve behavior at the points

where it matters most. Over time, this enables systems to become both more capable and easier to use.

Finally, an important direction is to extend ideas from test-time scaling and test-time training to embodied agents. Additional computation at inference time, such as reasoning over candidate plans or refining behavior through interaction, can improve performance without requiring new data collection. At the same time, updating internal representations or policies during execution allows systems to adapt within a single deployment, blurring the boundary between inference and learning.

These directions point toward robot systems that are not only pre-trained, but continually improving. Human input provides the initial guidance, while interaction and experience drive ongoing refinement. This combination is essential for achieving reliable performance in open-world environments.

BIBLIOGRAPHY

Bibliography

- [1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.
- [2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.
- [3] Ademi Adeniji, Amber Xie, Carmelo Sferrazza, Younggyo Seo, Stephen James, and Pieter Abbeel. Language reward modulation for pretraining reinforcement learning. *arXiv preprint arXiv:2308.12270*, 2024.
- [4] Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.
- [5] Christopher Agia, Rohan Sinha, Jingyun Yang, Rika Antonova, Marco Pavone, Haruki Nishimura, Masha Itkina, and Jeannette Bohg. Cupid: Curating data your robot loves with influence functions. In *Conference on Robot Learning (CoRL)*, 2025.
- [6] Christopher Agia, Rohan Sinha, Jingyun Yang, Ziang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In *Conference on Robot Learning (CoRL)*, 2025.
- [7] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.
- [8] Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.
- [9] Akanksha Atrey, Kaleigh Clary, and David Jensen. Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning. *International Conference on Learning Representations (ICLR)*, 2020.
- [10] Pranav Atreya, Karl Pertsch, Tony Lee, Moo Jin Kim, Arhan Jain, Artur Kuramshin, Clemens Eppner, Cyrus Neary, Edward Hu, Fabio Ramos, et al. Roboarena: Distributed real-world evaluation of generalist robot policies. In *Conference on Robot Learning (CoRL)*, 2025.

- [11] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- [12] Roman Bachmann, David Mizrahi, Andrei Atanov, and Amir Zamir. MultiMAE: Multi-modal Multi-task Masked Autoencoders. *European Conference on Computer Vision (ECCV)*, 2022.
- [13] Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [14] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibao Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.
- [15] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.
- [16] Andrea Bajcsy, Dylan P Losey, Marcia K O’Malley, and Anca D Dragan. Learning from physical human corrections, one feature at a time. In *International Conference on Human-Robot Interaction (HRI)*, 2018.
- [17] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [18] Anton Bakhtin, Yuntian Deng, Sam Gross, Myle Ott, Marc’Aurelio Ranzato, and Arthur Szlam. Residual energy-based models for text. *Journal of Machine Learning Research*, 2021.
- [19] Yuwei Bao, Sayan Ghosh, and Joyce Chai. Learning to mediate disparities towards pragmatic communication. *arXiv preprint arXiv:2203.13685*, 2022.
- [20] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa M. Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.
- [21] Jacob Beck, Risto Vuorio, Zheng Xiong, and Shimon Whiteson. Recurrent hypernetworks are surprisingly strong in meta-rl. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

- [22] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. Hydra: Hybrid robot actions for imitation learning. In *Conference on Robot Learning (CoRL)*, 2023.
- [23] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.
- [24] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 1957.
- [25] Luciana Benotti and Patrick Rowan Blackburn. Grounding as a collaborative process. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, 2021.
- [26] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *International Conference on Machine Learning (ICML)*, 2021.
- [27] David Bertoin, Adil Zouitine, Mehdi Zouitine, and Emmanuel Rachelson. Look where you look! Saliency-guided Q-networks for generalization in visual Reinforcement Learning. *Neural Information Processing Systems (NeurIPS)*, 2022.
- [28] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*. 2012.
- [29] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. *arXiv preprint arXiv:2309.01918*, 2023.
- [30] Erdem Biyik, Jonathan Margoliash, Shahrouz Ryan Alimo, and Dorsa Sadigh. Efficient and safe exploration in deterministic markov decision processes with unknown transition models. In *American Control Conference*, 2019.
- [31] Erdem Biyik, Nicolas Huynh, Mykel J. Kochenderfer, and Dorsa Sadigh. Active preference-based gaussian process regression for reward learning. In *Robotics: Science and Systems (RSS)*, 2020.
- [32] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arxiv:2410.24164*, 2024.
- [33] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.
- [34] Andreea Bobu, Marius Wiggert, Claire Tomlin, and Anca D Dragan. Feature Expansive Reward Learning: Rethinking Human Input. In *International Conference on Human-Robot Interaction (HRI)*, 2021.

- [35] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- [36] Aidan Boyd, Kevin W Bowyer, and Adam Czajka. Human-Aided Saliency Maps Improve Generalization of Deep Learning. In *Winter Conference on Applications of Computer Vision (WACV)*, 2022.
- [37] Aidan Boyd, Patrick Tinsley, Kevin W Bowyer, and Adam Czajka. CYBORG: Blending Human Saliency Into the Loss Improves Deep Learning. In *Winter Conference on Applications of Computer Vision (WACV)*, 2023.
- [38] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 1952.
- [39] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [40] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.
- [41] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems (RSS)*, 2023.
- [42] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Maria Elisabeth Bechtle, Feryal Behbahani, Stephanie C.Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments. In *International Conference on Machine Learning (ICML)*, 2024.
- [43] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Learning to act anywhere with task-centric latent actions. *Robotics: Science and Systems*, 2025.
- [44] Paweł Budzianowski, Emilia Wiśnios, Gracjan Góral, Igor Kulakov, Viktor Petrenko, and Krzysztof Walas. Opengvl: Benchmarking visual temporal progress for data curation. *arXiv preprint arXiv:2509.17321*, 2025.

- [45] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *Robotics: Science and Systems (RSS)*, 2020.
- [46] Zhangjie Cao, Erdem Biyik, Woodrow Z Wang, Allan Raventos, Adrien Gaidon, Guy Rosman, and Dorsa Sadigh. Reinforcement learning based control of imitative policies for near-accident driving. *Robotics: Science and Systems*, 2020.
- [47] Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Fred Shentu, and Philipp Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. *arXiv preprint arXiv:2509.25358*, 2025.
- [48] Shirui Chen, Cole Harrison, Ying-Chun Lee, Angela Jin Yang, Zhongzheng Ren, Lillian J. Ratliff, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Topreward: Token probabilities as hidden zero-shot rewards for robotics. *arXiv preprint arXiv:2602.19313*, 2026.
- [49] Xiaoyu Chen, Xiangming Zhu, Yufeng Zheng, Pushi Zhang, Li Zhao, Wenxue Cheng, Peng Cheng, Yongqiang Xiong, Tao Qin, Jianyu Chen, et al. An adaptive deep rl method for non-stationary environments with piecewise stable context. *Neural Information Processing Systems*, 2022.
- [50] Xiaoyu Chen, Hangxing Wei, Pushi Zhang, Chuheng Zhang, Kaixin Wang, Yanjiang Guo, Rushuai Yang, Yucen Wang, Xinquan Xiao, Li Zhao, Jianyu Chen, and Jiang Bian. villa-x: Enhancing latent action modeling in vision-language-action models. *arXiv preprint arXiv:2507.23682*, 2025.
- [51] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for $2+3=?$ on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.
- [52] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [53] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.
- [54] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
- [55] Era Choshen and Aviv Tamar. Contrabar: Contrastive bayes-adaptive deep rl. In *International Conference on Machine Learning (ICML)*, 2023.

- [56] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [57] Christopher Clark, Jieyu Zhang, Zixian Ma, Jae Sung Park, Mohammadreza Salehi, Rohun Tripathi, Sangho Lee, Zhongzheng Ren, Chris Dongjoo Kim, Yinuo Yang, Vincent Shao, Yue Yang, Weikai Huang, Ziqi Gao, Taira Anderson, Jianrui Zhang, Jitesh Jain, George Stoica, Winson Han, Ali Farhadi, and Ranjay Krishna. Molmo2: Open weights and data for vision-language models with video understanding and grounding. *arXiv preprint arXiv:2601.10611*, 2026.
- [58] Herbert H Clark and Edward F Schaefer. Contributing to discourse. *Cognitive science*, 1989.
- [59] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning (CoRL)*, 2018.
- [60] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.
- [61] Open X-Embodiment Collaboration et al. Open X-Embodiment: Robotic learning datasets and RT-X models, 2023.
- [62] AgiBot World Colosseum contributors. Agibot world colosseum, 2024.
- [63] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.
- [64] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for Dynamics and Control Conference (L4DC)*, 2022.
- [65] Zichen Jeff Cui, Hengkai Pan, Aadithya Iyer, Siddhant Haldar, and Lerrel Pinto. Dynamo: In-domain dynamics pretraining for visuo-motor control. In *International Conference on Machine Learning (ICML)*, 2024.
- [66] Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. Racer: Rich language-guided failure recovery policies for imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [67] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Jian Ma, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100. *International Journal of Computer Vision (IJCV)*, 2022.
- [68] Kevin P Darby, Sophia W Deng, Dirk B Walther, and Vladimir M Sloutsky. The development of attention to objects and scenes: From object-biased to unbiased. *Child development*, 2021.

- [69] Abhishek Das, Harsh Agrawal, Larry Zitnick, Devi Parikh, and Dhruv Batra. Human Attention in Visual Question Answering: Do Humans and Deep Networks Look at the Same Regions? *Computer Vision and Image Understanding*, 2017.
- [70] Shivin Dass, Jullian Yapeter, Jesse Zhang, Jiahui Zhang, Karl Pertsch, Stefanos Nikolaidis, and Joseph J. Lim. Clvr jaco play dataset, 2023.
- [71] Matt Deitke et al. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146*, 2024.
- [72] Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta reinforcement learning—identifiability challenges and effective data collection strategies. *Neural Information Processing Systems*, 2021.
- [73] Ria Doshi, Homer Rich Walke, Oier Mees, Sudeep Dasari, and Sergey Levine. Scaling cross-embodied learning: One policy for manipulation, navigation, locomotion and aviation. In *Conference on Robot Learning (CoRL)*, 2024.
- [74] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations (ICLR)*, 2021.
- [75] Maximilian Du, Suraj Nair, Dorsa Sadigh, and Chelsea Finn. Behavior retrieval: Few-shot imitation learning by querying unlabeled datasets. In *Robotics: Science and Systems (RSS)*, 2023.
- [76] Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [77] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2023.
- [78] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL²: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.
- [79] Michael O’Gordon Duff. *Optimal learning: computational procedures for bayes-adaptive markov decision processes*. PhD thesis, 2002.
- [80] Zackory Erickson, Vamsee Gangaram, Ariel Kapusta, C Karen Liu, and Charles C Kemp. Assistive gym: A physics simulation framework for assistive robotics. In *International Conference on Robotics and Automation (ICRA)*, 2020.
- [81] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*, 2018.

- [82] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems (NeurIPS): Datasets and Benchmarks Track*, 2022.
- [83] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.
- [84] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taiga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep RL. In *International Conference on Machine Learning (ICML)*, 2024.
- [85] Constanza Fierro, Reinald Kim Amplayo, Fantine Huot, Nicola De Cao, Joshua Maynez, Shashi Narayan, and Mirella Lapata. Learning to plan and generate text with citations. *arXiv preprint arXiv:2404.03381*, 2024.
- [86] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*, 2016.
- [87] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.
- [88] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.
- [89] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.
- [90] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [91] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.
- [92] Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. Robot policy learning with temporal optimal transport reward. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [93] Yuwei Fu, Haichao Zhang, Di Wu, Wei Xu, and Benoit Boulet. FuRL: Visual-language models as fuzzy rewards for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2024.

- [94] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.
- [95] Shenyuan Gao, William Liang, Kaiyuan Zheng, Ayaan Malik, Seonghyeon Ye, Sihyun Yu, Wei-Cheng Tseng, Yuzhu Dong, Kaichun Mo, Chen-Hsuan Lin, Qianli Ma, Seungjun Nah, Loic Magne, Jiannan Xiang, Yuqi Xie, Ruijie Zheng, Dantong Niu, You Liang Tan, K.R. Zentner, George Kurian, Suneel Indupuru, Pooya Jannaty, Jinwei Gu, Jun Zhang, Jitendra Malik, Pieter Abbeel, Ming-Yu Liu, Yuke Zhu, Joel Jang, and Linxi "Jim" Fan. Dreamdojo: A generalist robot world model from large-scale human videos. *arXiv preprint arXiv:2602.06949*, 2026.
- [96] Quentin Garrido, Tushar Nagarajan, Basile Terver, Nicolas Ballas, Yann LeCun, and Michael Rabbat. Learning latent action world models in the wild, 2026.
- [97] Seyed Kamyar Seyed Ghasemipour, Ayzaan Wahid, Jonathan Tompson, Pannag R Sanketi, and Igor Mordatch. Self-improving embodied foundation models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [98] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 2015.
- [99] Ankit Goyal, Valts Blukis, Jie Xu, Yijie Guo, Yu-Wei Chao, and Dieter Fox. Rvt2: Learning precise manipulation from few demonstrations. *Robotics: Science and Systems*, 2024.
- [100] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, Miguel Martin, Tushar Nagarajan, Ilija Radosavovic, Santhosh Kumar Ramakrishnan, Fiona Ryan, Jayant Sharma, Michael Wray, Mengmeng Xu, Eric Zhongcong Xu, Chen Zhao, Siddhant Bansal, Dhruv Batra, Vincent Cartillier, Sean Crane, Tien Do, Morrie Doulaty, Akshay Erapalli, Christoph Feichtenhofer, Adriano Fragomeni, Qichen Fu, Abrahm Gebreselasie, Cristina González, James Hillis, Xuhua Huang, Yifei Huang, Wenqi Jia, Weslie Khoo, Jáchym Kolář, Satwik Kottur, Anurag Kumar, Federico Landini, Chao Li, Yanghao Li, Zhenqiang Li, Karttikeya Mangalam, Raghava Modhugu, Jonathan Munro, Tullie Murrell, Takumi Nishiyasu, Will Price, Paola Ruiz Puentes, Merey Ramazanova, Leda Sari, Kiran Soma-sundaram, Audrey Southerland, Yusuke Sugano, Ruijie Tao, Minh Vo, Yuchen Wang, Xindi Wu, Takuma Yagi, Ziwei Zhao, Yunyi Zhu, Pablo Arbeláez, David Crandall, Dima Damen, Giovanni Maria Farinella, Christian Fuegen, Bernard Ghanem, Vamsi Krishna Ithapu, C. V. Jawahar, Hanbyul Joo, Kris Kitani, Haizhou Li, Richard Newcombe, Aude Oliva, Hyun Soo Park, James M. Rehg, Yoichi Sato, Jianbo Shi, Mike Zheng Shou, Antonio Torralba, Lorenzo Torresani, Mingfei Yan, and Jitendra Malik. Ego4d: Around the world in 3,000 hours of ego-centric video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [101] Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. SAFE: Multitask failure detection for vision-language-action models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

- [102] Lin Guan, Yifan Zhou, Denis Liu, Yantian Zha, Heni Ben Amor, and Subbarao Kambhampati. Task success is not enough: Investigating the use of video-language models as behavior critics for catching undesirable agent behaviors. In *Conference on Language Modeling (COLM)*, 2024.
- [103] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [104] Alexander Gurung and Mirella Lapata. Learning to reason for long-form story generation. *arXiv preprint arXiv:2503.22828*, 2025.
- [105] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.
- [106] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft Actor-Critic Algorithms and Applications. *International Conference on Machine Learning (ICML)*, 2018.
- [107] Siddhant Halder and Lerrel Pinto. Point policy: Unifying observations and actions with key points for robot manipulation. *arXiv preprint arXiv:2502.20391*, 2025.
- [108] Siddhant Halder, Vaibhav Mathur, Denis Yarats, and Lerrel Pinto. Watch and match: Supercharging imitation with regularized optimal transport. In *Conference on Robot Learning (CoRL)*, 2023.
- [109] Siddhant Halder, Zhuoran Peng, and Lerrel Pinto. Baku: An efficient transformer for multi-task policy learning. *Neural Information Processing Systems*, 2024.
- [110] Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*, 2024.
- [111] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *International Conference on Robotics and Automation (ICRA)*, 2021.
- [112] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning (ICML)*, 2022.
- [113] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.
- [114] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked Autoencoders Are Scalable Vision Learners. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [115] Joey Hejna and Dorsa Sadigh. Few-shot preference learning for human-in-the-loop rl. In *Conference on Robot Learning (CoRL)*, 2022.

- [116] Joey Hejna, Suvir Mirchandani, Ashwin Balakrishna, Annie Xie, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, Dhruv Shah, Coline Devin, and Dorsa Sadigh. Robot data curation with mutual information estimators. In *Robotics: Science and Systems (RSS)*, 2025.
- [117] Minh Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems (RSS)*, 2023.
- [118] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [119] Matthew Hong, Anthony Liang, Kevin Kim, Harshitha Rajaprakash, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Hand me the data: Fast robot adaptation via hand path retrieval, 2025.
- [120] Matthew Hong, Anthony Liang, Kevin Kim, Harshitha Rajaprakash, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Hand me the data: Fast robot adaptation via hand path retrieval. In *International Conference on Robotics and Automation (ICRA)*, 2026.
- [121] Ryan Hoque, Peide Huang, David J. Yoon, Mouli Sivapurapu, and Jian Zhang. Egodex: Learning dexterous manipulation from large-scale egocentric video. *arXiv preprint arXiv:2505.11709*, 2025.
- [122] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022.
- [123] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *International Conference on Learning Representations (ICLR) Blog Track*, 2022.
- [124] Yuhang Huang, Shilong Zou, Jiazhao Zhang, Xinwang Liu, Ruizhen Hu, and Kai Xu. Adapower: Specializing world foundation models for predictive manipulation. *arXiv preprint arXiv:2512.035358*, 2025.
- [125] Yuheng Huang, Jiayang Song, Zhijie Wang, Shengming Zhao, Huaming Chen, Felix Juefei-Xu, and Lei Ma. Look before you leap: An exploratory study of uncertainty measurement for large language models. *arXiv preprint arXiv:2307.10236*, 2023.
- [126] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.
- [127] Kuo-Han Hung, Pang-Chi Lo, Jia-Fong Yeh, Han-Yuan Hsu, Yi-Ting Chen, and Winston H. Hsu. VICtor: Learning hierarchical vision-instruction correlation rewards for long-horizon manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

- [128] Fantine Huot, Joshua Maynez, Shashi Narayan, Reinald Kim Amplayo, Kuzman Ganchev, Annie Priyadarshini Louis, Anders Sandholm, Dipanjan Das, and Mirella Lapata. Text-blueprint: An interactive platform for plan-based conditional generation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2023.
- [129] Minyoung Hwang, Alexandra Forsey-Smerek, Nathaniel Dennler, and Andreea Bobu. Masked irl: Llm-guided reward disambiguation from demonstrations and language. *arXiv preprint arXiv:2511.14565*, 2025.
- [130] Minyoung Hwang, Joey Hejna, Dorsa Sadigh, and Yonatan Bisk. Motif: Motion instruction fine-tuning. *IEEE Robotics and Automation Letters (RA-L)*, 2025.
- [131] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. RecSim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019.
- [132] Arda Inceoglu, Eren Erdal Aksoy, Abdullah Cihan Ak, and Sanem Sariel. Fino-net: A deep multimodal sensor fusion framework for manipulation failure detection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.
- [133] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi_{0.6}^*$: A vla that learns from experience. *arXiv:2511.14759*, 2025.
- [134] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [135] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.
- [136] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

- [137] Arnav Kumar Jain, Vibhakar Mohta, Subin Kim, Atiksh Bhardwaj, Juntao Ren, Yunhai Feng, Sanjiban Choudhury, and Gokul Swamy. A smooth sea never made a skilled SAILOR: Robust imitation via learning to search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [138] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.
- [139] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning (CoRL)*, 2021.
- [140] Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. A survey on conversational recommender systems. *ACM Computing Surveys (CSUR)*, 2021.
- [141] Gawesh Jawaheer, Peter Weller, and Patty Kostkova. Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. *ACM Transactions on Interactive Intelligent Systems*, 2014.
- [142] Tao Jiang, Tianyuan Yuan, Yicheng Liu, Chenhao Lu, Jianning Cui, Xiao Liu, Shuiqi Cheng, Jiyang Gao, Huazhe Xu, and Hang Zhao. Galaxea open-world dataset and g0 dual-system vla model. *arXiv preprint arXiv:2509.00576*, 2025.
- [143] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 1998.
- [144] Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2025.
- [145] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. In *European Conference on Computer Vision (ECCV)*, 2025.
- [146] Siddharth Karamcheti, Suraj Nair, Annie S Chen, Thomas Kollar, Chelsea Finn, Dorsa Sadigh, and Percy Liang. Language-Driven Representation Learning for Robotics. *arXiv preprint arXiv:2302.12766*, 2023.
- [147] Simar Kareer, Dhruv Patel, Ryan Punamiya, Pranay Mathur, Shuo Cheng, Chen Wang, Judy Hoffman, and Danfei Xu. Egomimic: Scaling imitation learning via egocentric video. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [148] Kushal Kedia, Prithwish Dan, Angela Chao, Maximus Adrian Pace, and Sanjiban Choudhury. One-shot imitation under mismatched execution. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [149] Alexander Khazatsky et al. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

- [150] Taylor W Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. *Neural Information Processing Systems*, 2017.
- [151] Changyeon Kim, Jongjin Park, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Preference transformer: Modeling human preferences using transformers for rl. *International Conference of Learning Representations*, 2023.
- [152] Changyeon Kim, Minhho Heo, Doohyun Lee, Honglak Lee, Jinwoo Shin, Joseph J. Lim, and Kimin Lee. Subtask-aware visual reward learning from segmented demonstrations. In *International Conference on Learning Representations (ICLR)*, 2025.
- [153] Minbeom Kim, Thibaut Thonet, Jos Rozen, Hwaran Lee, Kyomin Jung, and Marc Dymetman. Guaranteed generation from large language models. *arXiv preprint arXiv:2410.06716*, 2024.
- [154] Moo Jin Kim, Jiajun Wu, and Chelsea Finn. Giving robots a hand: Learning generalizable manipulation with eye-in-hand human video demonstrations. *CoRR*, 2023.
- [155] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.
- [156] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- [157] Albina Klepach, Alexander Nikulin, Ilya Zisman, Denis Tarasov, Alexander Derevyagin, Andrei Polubarov, Nikita Lyubaykin, and Vladislav Kurenkov. Object-centric latent action learning. *arXiv preprint arXiv:2502.09680*, 2025.
- [158] Po-Chen Ko, Jiayuan Mao, Yilun Du, Shao-Hua Sun, and Joshua B. Tenenbaum. Learning to act from actionless videos through dense correspondences. In *International Conference on Learning Representations (ICLR)*, 2024.
- [159] Yigit Korkmaz and Erdem Biyik. Mile: Model-based intervention learning. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [160] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2021.
- [161] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.
- [162] Kalpesh Krishna, Yapei Chang, John Wieting, and Mohit Iyyer. Rankgen: Improving text generation with large ranking models. *arXiv preprint arXiv:2205.09726*, 2022.
- [163] Yuxuan Kuang, Junjie Ye, Haoran Geng, Jiageng Mao, Congyue Deng, Leonidas Guibas, He Wang, and Yue Wang. Ram: Retrieval-based affordance transfer for generalizable zero-shot robotic manipulation. *Conference on Robot Learning*, 2024.

- [164] Matthias Kümmerer, Thomas SA Wallis, and Matthias Bethge. DeepGaze II: Reading fixations from deep features trained on object recognition. *arXiv preprint arXiv:1610.01563*, 2016.
- [165] Matthias Kümmerer, Matthias Bethge, and Thomas SA Wallis. Deepgaze iii: Modeling free-viewing human scanpaths with deep learning. *Journal of Vision*, 2022.
- [166] Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. In *Conference on Robot Learning (CoRL)*, 2025.
- [167] Jeongyeol Kwon, Yonathan Efroni, Constantine Caramanis, and Shie Mannor. RL for latent mdps: Regret guarantees and a lower bound. *Neural Information Processing Systems*, 2021.
- [168] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [169] D. R. J. Laming. The relativity of ‘absolute’ judgements. *British Journal of Mathematical and Statistical Psychology*, 1984.
- [170] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2020.
- [171] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement Learning with Augmented Data. *Neural Information Processing Systems (NeurIPS)*, 2020.
- [172] Gilwoo Lee, Brian Hou, Aditya Mandalika, Jeongseok Lee, Sanjiban Choudhury, and Siddhartha S Srinivasa. Bayesian policy optimization for model uncertainty. *International Conference on Learning Representations*, 2018.
- [173] Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. Molmoact: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.
- [174] Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. In *International Conference on Machine Learning (ICML)*, 2021.
- [175] Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. Roboreward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.
- [176] Marion Lepert, Jiaying Fang, and Jeannette Bohg. Masquerade: Learning from in-the-wild human videos using data-editing. *arXiv preprint arXiv:2508.09976*, 2025.
- [177] Marion Lepert, Jiaying Fang, and Jeannette Bohg. Phantom: Training robots without robots using only human videos. In *Conference on Robot Learning (CoRL)*, 2025.

- [178] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, 2023.
- [179] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [180] Haozhuo Li, Yuchen Cui, and Dorsa Sadigh. How to train your robots? the impact of demonstration modality on imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [181] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 2023.
- [182] Qingshan Li, Yue Zhou, and Jie Yang. Saliency Based Image Segmentation. In *International Conference on Information and Multimedia Technology (ICIMT)*, 2011.
- [183] Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. *arXiv preprint arXiv:2210.15097*, 2022.
- [184] Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. Hamster: Hierarchical action models for open-world robot manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.
- [185] Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical action models for open-world robot manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.
- [186] Anthony Liang, Ishika Singh, Karl Pertsch, and Jesse Thomason. Transformer adapters for robot learning. In *CoRL 2022 Workshop on Pre-training Robot Learning*, 2022.
- [187] Anthony Liang, Chih-wei Hsu, Yinlam Chow, Guy Tennenholtz, Erdem Biyik, and Craig Boutilier. DynaMITE-RL: A dynamic model for improved temporal meta-reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [188] Anthony Liang, Jesse Thomason, and Erdem Biyik. Visarl: Visual reinforcement learning guided by human saliency. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.
- [189] Anthony Liang, Pavel Czempin, Matthew Hong, Yutai Zhou, Erdem Biyik, and Stephen Tu. CLAM: Continuous latent action models for robot learning from unlabeled demonstrations. *arXiv preprint arXiv:2505.04999*, 2025.

- [190] Anthony Liang, Kalpesh Krishna, Abhimanyu Goyal, Jacob Eisenstein, Adam Fisch, and Jonathan Berant. Inference-time plan guidance with interleaved reasoning. In *International Conference on Machine Learning (ICML)*, 2026. Spotlight.
- [191] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- [192] Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.
- [193] Fanqi Lin, Kushal Arora, Jean Mercat, Haruki Nishimura, Paarth Shah, Chen Xu, Mengchao Zhang, Mark Zolotas, Maya Angeles, Owen Pfannenstiehl, Andrew Beaulieu, and Jose Barreiros. A systematic study of data modalities and strategies for co-training large behavior models for robot manipulation, 2026.
- [194] Li-Heng Lin, Yuchen Cui, Amber Xie, Tianyu Hua, and Dorsa Sadigh. Flowretrieval: Flow-guided data retrieval for few-shot imitation learning. In *Conference on Robot Learning (CoRL)*, 2024.
- [195] Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bihan Wen. Failsafe: Reasoning and recovery from failures in vision-language-action models. *arXiv preprint arXiv:2510.01642*, 2025.
- [196] Akis Linardos, Matthias Kümmerer, Ori Press, and Matthias Bethge. DeepGaze IIE: Calibrated prediction in and out-of-domain for state-of-the-art saliency modeling. In *International Conference on Computer Vision (ICCV)*, 2021.
- [197] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Neural Information Processing Systems Track Datasets and Benchmarks*, 2023.
- [198] Fangchen Liu, Zhan Ling, Tongzhou Mu, and Hao Su. State alignment-based imitation learning. In *International Conference on Learning Representations (ICLR)*, 2020.
- [199] Nian Liu, Junwei Han, and Ming-Hsuan Yang. PiCANet: Learning Pixel-wise Contextual Attention for Saliency Detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [200] Siqi Liu, Kay Choong See, Kee Yuan Ngiam, Leo Anthony Celi, Xingzhi Sun, and Mengling Feng. Reinforcement learning for clinical decision support in critical care: comprehensive review. *Journal of Medical Internet Research*, 2020.
- [201] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *International Conference on Robotics and Automation (ICRA)*, 2018.

- [202] Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. TAIL: Task-specific adapters for imitation learning with large pretrained models. In *International Conference on Learning Representations (ICLR)*, 2024.
- [203] Weifeng Lu, Minghao Ye, Zewei Ye, Ruihan Tao, Shuo Yang, and Bo Zhao. Robofac: A comprehensive framework for robotic failure analysis and correction. *arXiv preprint arXiv:2505.12224*, 2025.
- [204] Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*, 2025.
- [205] Jianlan Luo, Charles Xu, Xinyang Geng, Gilbert Feng, Kuan Fang, Liam Tan, Stefan Schaal, and Sergey Levine. Multi-stage cable routing through hierarchical imitation learning. *arXiv preprint arXiv:2307.08927*, 2023.
- [206] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *Conference on Robot Learning (CoRL)*, 2019.
- [207] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning (CoRL)*, 2020.
- [208] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters (RA-L)*, 2023.
- [209] Yecheng Jason Ma, William Liang, Vaidehi Som, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *International Conference on Machine Learning (ICML)*, 2023.
- [210] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations (ICLR)*, 2023.
- [211] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.
- [212] Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Jonathan Tompson, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners. In *International Conference on Learning Representations (ICLR)*, 2025.

- [213] Parsa Mahmoudieh, Deepak Pathak, and Trevor Darrell. Zero-shot reward specification via grounded natural language. In *International Conference on Machine Learning (ICML)*, 2022.
- [214] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *Robotics and Automation Letters (RA-L)*, 2022.
- [215] Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. STRAP: Robot sub-trajectory retrieval for augmented policy learning. In *International Conference on Learning Representations (ICLR)*, 2025.
- [216] Willi Menapace, Stephane Lathuiliere, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [217] Chancharik Mitra, Abrar Anwar, Rodolfo Corona, Dan Klein, Trevor Darrell, and Jesse Thomason. Which one? leveraging context between objects and multiple views for language grounding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.
- [218] Will Monroe, Robert XD Hawkins, Noah Goodman, and Christopher Potts. Colors in context: A pragmatic neural model for grounded language understanding. *Transactions of the Association for Computational Linguistics (TACL)*, 2017.
- [219] Meinard Müller. *Fundamentals of music processing: Using Python and Jupyter notebooks*. 2021.
- [220] T Nathan Mundhenk, Barry Y Chen, and Gerald Friedland. Efficient Saliency Maps for Explainable AI. *arXiv preprint arXiv:1911.11293*, 2019.
- [221] Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*, 2025.
- [222] Calarina Muslimani, Kerrick Johnstonbaugh, Suyog Chandramouli, Serena Booth, W. Bradley Knox, and Matthew E. Taylor. Towards improving reward design in RL: A reward alignment metric for RL practitioners. In *Reinforcement Learning Conference (RLC)*, 2025.
- [223] Vivek Myers, Erdem Biyik, Nima Anari, and Dorsa Sadigh. Learning multimodal rewards from rankings. In *Conference on Robot Learning (CoRL)*, 2021.
- [224] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3M: A Universal Visual Representation for Robot Manipulation. *Conference on Robot Learning (CoRL)*, 2022.

- [225] Taewook Nam, Juyong Lee, Jesse Zhang, Sung Ju Hwang, Joseph J. Lim, and Karl Pertsch. Lift: Unsupervised reinforcement learning with foundation models as teachers. *arXiv preprint arXiv:2312.08958*, 2023.
- [226] Soroush Nasiriany, Tian Gao, Ajay Mandlekar, and Yuke Zhu. Learning and retrieval from prior data for skill-based imitation learning. In *Conference on Robot Learning (CoRL)*, 2022.
- [227] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.
- [228] Alexander Nikulin, Ilya Zisman, Denis Tarasov, Lyubaykin Nikita, Andrei Polubarov, Igor Kiselev, and Vladislav Kurenkov. Latent action learning requires supervision in the presence of distractors. In *International Conference on Machine Learning (ICML)*, 2025.
- [229] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep?: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning (ICML)*, 2023.
- [230] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems (RSS)*, 2024.
- [231] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.
- [232] OpenAI, 2025.
- [233] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khali-dov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision, 2024.
- [234] Takayuki Osa. Motion planning by learning the solution manifold in trajectory optimization. *The International Journal of Robotics Research*, 2022.
- [235] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. Quality: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608*, 2021.
- [236] Dim P Papadopoulos, Alasdair DF Clarke, Frank Keller, and Vittorio Ferrari. Training Object Class Detectors from Eye Tracking Data. In *European Conference on Computer Vision (ECCV)*, 2014.

- [237] Georgios Papagiannis, Norman Di Palo, Pietro Vitiello, and Edward Johns. R+ x: Retrieval and execution from everyday human videos. In *International Conference on Robotics and Automation (ICRA)*, 2025.
- [238] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.
- [239] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.
- [240] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.
- [241] Ri-Zhao Qiu, Shiqi Yang, Xuxin Cheng, Chaitanya Chawla, Jialong Li, Tairan He, Ge Yan, David J Yoon, Ryan Hoque, Lars Paulsen, et al. Humanoid policy~ human policy. *arXiv preprint arXiv:2503.13441*, 2025.
- [242] Gabriel Quere, Annette Hagenhuber, Maged Iskandar, Samuel Bustamante, Daniel Leidner, Freek Stulp, and Joern Vogel. Shared Control Templates for Assistive Robotics. In *International Conference on Robotics and Automation (ICRA)*, 2020.
- [243] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning (CoRL)*, 2022.
- [244] Ilija Radosavovic, Baifeng Shi, Letian Fu, Ken Goldberg, Trevor Darrell, and Jitendra Malik. Robot learning with sensorimotor pre-training. *arXiv preprint arXiv:2306.10007*, 2023.
- [245] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning (ICML)*, 2019.
- [246] Mrinal Rawat, Ambuje Gupta, Rushil Goomer, Alessandro Di Bari, Neha Gupta, and Roberto Pieraccini. Pre-act: Multi-step planning and reasoning improves acting in llm agents. *arXiv preprint arXiv:2505.09970*, 2025.
- [247] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.
- [248] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.
- [249] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive pomdps. *Neural Information Processing Systems*, 2007.

- [250] Matthias Rosynski, Frank Kirchner, and Matias Valdenegro-Toro. Are Gradient-based Saliency Maps Useful in Deep Reinforcement Learning? *arXiv preprint arXiv:2012.01281*, 2020.
- [251] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Prompt: Proximal meta-policy search. *International Conference on Learning Representations*, 2018.
- [252] Dmitry Rudoy, Dan B Goldman, Eli Shechtman, and Lihi Zelnik-Manor. Learning video saliency from human gaze using candidate selection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [253] Dorsa Sadigh, Anca D. Dragan, S. Shankar Sastry, and Sanjit A. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*, 2017.
- [254] Alexander Sax, Bradley Emi, Amir R Zamir, Leonidas Guibas, Silvio Savarese, and Jitendra Malik. Mid-Level Visual Representations Improve Generalization and Sample Efficiency for Learning Visuomotor Policies. *Conference on Robot Learning (CoRL)*, 2019.
- [255] Saumya Saxena, Mohit Sharma, and Oliver Kroemer. Multi-resolution sensing for real-time control with vision-language models. In *Conference on Robot Learning (CoRL)*, 2023.
- [256] Dominik Schmidt and Minqi Jiang. Learning to act without actions. In *International Conference on Learning Representations (ICLR)*, 2023.
- [257] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [258] Rutav Shah, Roberto Martín-Martín, and Yuke Zhu. MUTEX: Learning unified policies from multimodal task specifications. In *Conference on Robot Learning (CoRL)*, 2023.
- [259] Omar Shaikh, Kristina Gligorić, Ashna Khetan, Matthias Gerstgrasser, Diyi Yang, and Dan Jurafsky. Grounding gaps in language model generations. *arXiv preprint arXiv:2311.09144*, 2023.
- [260] Marissa A. Sharif and Daniel M. Oppenheimer. The effect of relative encoding on memory-based judgments. *Psychological Science*, 2016.
- [261] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, 2004.
- [262] Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *arXiv preprint arXiv:2503.04472*, 2025.
- [263] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

- [264] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2023.
- [265] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv preprint arXiv:1312.6034*, 2013.
- [266] Sumedh Anand Sontakke, Jesse Zhang, Séb Arnold, Karl Pertsch, Erdem Biyik, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. Roboclip: One demonstration is enough to learn robot policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [267] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. REGENT: A retrieval-augmented generalist agent that can act in-context in new environments. In *International Conference on Learning Representations (ICLR)*, 2025.
- [268] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, and Insup Lee. Ricl: Adding in-context adaptability to pre-trained vision-language-action models. In *Conference on Robot Learning (CoRL)*, 2025.
- [269] Lauren N Steimle, David L Kaufman, and Brian T Denton. Multi-model markov decision processes. *IJSE Transactions*, 2021.
- [270] Neil Stewart, Gordon D. A. Brown, and Nick Chater. Absolute identification by relative judgment. *Psychological review*, 2005.
- [271] Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.
- [272] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018.
- [273] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2018.
- [274] Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, Shaoxuan Xie, Guocai Yao, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. *arXiv preprint arXiv:2512.23703*, 2025.
- [275] Stone Tao, Xiaochen Li, Tongzhou Mu, Zhiao Huang, Yuzhe Qin, and Hao Su. Abstract-to-Executable Trajectory Translation for One-Shot Task Generalization. In *Neural Information Processing Systems (NeurIPS) Deep Reinforcement Learning Workshop*, 2022.
- [276] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics

- simulation and rendering for generalizable embodied ai. In *Robotics: Science and Systems (RSS)*, 2025.
- [277] Gemini Team. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2024.
 - [278] Gemini Robotics Team et al. Gemini robotics: Bringing ai into the physical world, 2025.
 - [279] Guy Tennenholtz, Assaf Hallak, Gal Dalal, Shie Mannor, Gal Chechik, and Uri Shalit. On covariate shift of latent confounders in imitation and reinforcement learning. *International Conference of Learning Representations*, 2022.
 - [280] Guy Tennenholtz, Nadav Merlis, Lior Shani, Martin Mladenov, and Craig Boutilier. Reinforcement learning with history dependent dynamic contexts. In *International Conference on Machine Learning (ICML)*, 2023.
 - [281] Bahey Tharwat, Yara Nasser, Ali Abouzeid, and Ian Reid. Latent action pretraining through world modeling, 2025.
 - [282] Ran Tian, Yilin Wu, and Andrea Bacjsy. Position: Good embodied reward models need bad behavior data. Technical report, 2026.
 - [283] Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive inverse dynamics models are scalable learners for robotic manipulation. *International Conference on Learning Representations*, 2024.
 - [284] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.
 - [285] Yubing Tong, Hubert Konik, Faouzi Cheikh, and Alain Tremeau. Full Reference Image Quality Assessment Based on Saliency Map Analysis. *Journal of Imaging Science and Technology*, 2010.
 - [286] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqu Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and Tasks for Continuous Control. *Software Impacts*, 2020.
 - [287] Aaron Van Den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 2017.
 - [288] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
 - [289] Sreyas Venkataraman, Yufei Wang, Ziyu Wang, Navin Sriram Ravie, Zackory Erickson, and David Held. Real-world offline reinforcement learning from vision language model feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

- [290] Jörn Vogel, Annette Hagengruber, Maged Iskandar, Gabriel Quere, Ulrike Leipscher, Samuel Bustamante, Alexander Dietrich, Hannes Hoeppe, Daniel Leidner, and Alin Albu-Schäffer. Edan - an emg-controlled daily assistant to help people with physical disabilities. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [291] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2025.
- [292] Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, 2023.
- [293] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dhharshan Kumaran, and Matt Botvinick. Learning to reinforce learn. *arXiv preprint arXiv:1611.05763*, 2016.
- [294] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to Detect Salient Objects with Image-level Supervision. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [295] Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. *Neural Information Processing Systems*, 2024.
- [296] Wenguan Wang, Jianbing Shen, Fang Guo, Ming-Ming Cheng, and Ali Borji. Revisiting Video Saliency: A Large-scale Benchmark and a New Model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [297] Xuanhan Wang, Lianli Gao, Jingkuan Song, and Hengtao Shen. Beyond Frame-level CNN: Saliency-Aware 3-D CNN With LSTM for Video Action Recognition. *IEEE Signal Processing Letters*, 2016.
- [298] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. In *International Conference on Machine Learning (ICML)*, 2024.
- [299] Chuan Wen, Xingyu Lin, John So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point trajectory modeling for policy learning, 2023.
- [300] Nils Wilde, Erdem Biyik, Dorsa Sadigh, and Stephen L Smith. Learning Reward Functions from Scale Feedback. In *Conference on Robot Learning (CoRL)*, 2022.
- [301] Amber Xie, Rahul Chand, Dorsa Sadigh, and Joey Hejna. Data retrieval with importance weights for few-shot imitation learning. In *Conference on Robot Learning (CoRL)*, 2025.

- [302] Annie Xie and Chelsea Finn. Lifelong robotic reinforcement learning by retaining experiences. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2022.
- [303] Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst continual structured non-stationarity. In *International Conference on Machine Learning (ICML)*, 2021.
- [304] Jianan Xie, Zhen Xu, Jiayu Zeng, Yuyang Gao, and Kenji Hashimoto. Human–robot interaction using dynamic hand gesture for teleoperation of quadruped robots with a robotic arm. *Electronics*, 2025.
- [305] Roy Xie, David Qiu, Deepak Gopinath, Dong Lin, Yanchao Sun, Chong Wang, Saloni Potdar, and Bhuwan Dhingra. Interleaved reasoning for large language models via reinforcement learning. *arXiv preprint arXiv:2505.19640*, 2025.
- [306] Sicheng Xie, Haidong Cao, Zejia Weng, Zhen Xing, Haoran Chen, Shiwei Shen, Jiaqi Leng, Zuxuan Wu, and Yu-Gang Jiang. Human2robot: Learning robot actions from paired human-robot videos. *arXiv preprint arXiv:2502.16587*, 2025.
- [307] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.
- [308] Haoyu Xiong, Haoyuan Fu, Jieyi Zhang, Chen Bao, Qiang Zhang, Yongxi Huang, Wenqiang Xu, Animesh Garg, and Cewu Lu. Robotube: Learning household manipulation from human videos with simulated twin environments. In *Conference on Robot Learning (CoRL)*, 2023.
- [309] Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? Uncertainty-aware runtime failure detection for imitation learning policies. In *Robotics: Science and Systems (RSS)*, 2025.
- [310] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, and Dacheng Tao. Gmflow: Learning optical flow via global matching. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [311] Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. Flow as the cross-domain manipulation interface. In *Conference on Robot Learning (CoRL)*, 2024.
- [312] Ge Yan, Kris Wu, and Xiaolong Wang. Ucsd kitchens dataset, 2023.
- [313] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

- [314] Chao Yang, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Huaping Liu, Junzhou Huang, and Chuang Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [315] Daniel Yang, Davin Tjia, Jacob Berg, Dima Damen, Pulkit Agrawal, and Abhishek Gupta. Rank2reward: Learning shaped reward functions from passive video. In *International Conference on Robotics and Automation (ICRA)*, 2024.
- [316] Jiange Yang, Yansong Shi, Haoyi Zhu, Mingyu Liu, Kaijing Ma, Yating Wang, Gangshan Wu, Tong He, and Limin Wang. Como: Learning continuous latent motion from internet videos for scalable robot learning, 2025.
- [317] Zhaojing Yang, Miru Jun, Jeremy Tien, Stuart J. Russell, Anca Dragan, and Erdem Biyik. Trajectory improvement and reward learning from comparative language feedback. In *Conference on Robot Learning (CoRL)*, 2024.
- [318] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [319] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Latent action pretraining from videos. In *International Conference on Learning Representations (ICLR)*, 2024.
- [320] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.
- [321] Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*, 2025.
- [322] Sarah Young, Jyothish Pari, Pieter Abbeel, and Lerrel Pinto. Playful interactions for representation learning. In *International Conference on Intelligent Robots and Systems*, 2022.
- [323] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 2021.
- [324] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

- [325] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, 2020.
- [326] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humpalik, et al. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning (CoRL)*, 2023.
- [327] Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. Rankrag: Unifying context ranking with retrieval-augmented generation in llms. *Advances in Neural Information Processing Systems*, 2024.
- [328] Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. General flow as foundation affordance for scalable robot learning. *Conference on Robot Learning*, 2024.
- [329] Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint arXiv:2509.15937*, 2025.
- [330] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *International Conference on Computer Vision (ICCV)*, 2023.
- [331] Jesse Zhang, Minh Heo, Zuxin Liu, Erdem Biyik, Joseph J Lim, Yao Liu, and Rasool Fakoor. EXTRACT: Efficient policy learning by extracting transferrable robot skills from offline data. In *Conference on Robot Learning (CoRL)*, 2024.
- [332] Jesse Zhang, Marius Memmel, Kevin Kim, Dieter Fox, Jesse Thomason, Fabio Ramos, Erdem Biyik, Abhishek Gupta, and Anqi Li. Peek: Guiding and minimal image representations for zero-shot generalization of robot manipulation policies. In *International Conference on Robotics and Automation (ICRA)*, 2026.
- [333] Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. ReWiND: Language-guided rewards teach robot policies without new demonstrations. In *Conference on Robot Learning (CoRL)*, 2025.
- [334] Jiahui Zhang, Anthony Liang, Yusen Luo, Matthew Hong, Yutai Zhou, Abrar Anwar, Zhaojing Yang, Lee Kezar, Sumedh Sontakke, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *Robotics: Science and Systems (RSS)*, 2026.
- [335] Jianshu Zhang, Chengxuan Qian, Haosen Sun, Haoran Lu, Dingcheng Wang, Letian Xue, and Han Liu. Progresslm: Towards progress reasoning in vision-language models. *arXiv preprint arXiv:2601.15224*, 2026.
- [336] Yu Zhang, Yuqi Xie, Huihan Liu, Rutav Shah, Michael Wan, Linxi Fan, and Yuke Zhu. Scizor: Self-supervised data curation for large-scale imitation learning. In *International Conference on Robotics and Automation (ICRA)*, 2026.

- [337] Rui Zhao, Wanli Oyang, and Xiaogang Wang. Person Re-Identification by Saliency Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.
- [338] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems (RSS)*, 2023.
- [339] Tony Z. Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity, 2024.
- [340] Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. The majority is not always right: RL training for solution aggregation. *arXiv preprint arXiv:2509.06870*, 2025.
- [341] Zhenyu Zhao, Hongyi Jing, Xiawei Liu, Jiageng Mao, Abha Jha, Hanwen Yang, Rong Xue, Sergey Zakharov, Vitor Guizilini, and Yue Wang. Humanoid everyday: A comprehensive robotic dataset for open-world humanoid manipulation. *arXiv preprint arXiv:2510.08807*, 2025.
- [342] Gaoyue Zhou, Victoria Dean, Mohan Kumar Srirama, Aravind Rajeswaran, Jyothish Pari, Kyle Hatch, Aryan Jain, Tianhe Yu, Pieter Abbeel, Lerrel Pinto, Chelsea Finn, and Abhinav Gupta. Train offline, test online: A real robot learning benchmark. In *International Conference on Robotics and Automation (ICRA)*, 2023.
- [343] Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Rich Walke, Oier Mees, and Sergey Levine. Autonomous improvement of instruction following skills via foundation models. In *Conference on Robot Learning (CoRL)*, 2024.
- [344] Zhiyuan Zhou, Pranav Atreya, You Liang Tan, Karl Pertsch, and Sergey Levine. Autoeval: Autonomous evaluation of generalist robot manipulation policies in the real world. *arXiv preprint arXiv:2503.24278*, 2025.
- [345] Xinghao Zhu, Ran Tian, Chenfeng Xu, Mingyu Ding, Wei Zhan, and Masayoshi Tomizuka. Fanuc manipulation: A dataset for learning-based manipulation with fanuc mate 200id robot, 2023.
- [346] Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *IEEE Robotics and Automation Letters (RA-L)*, 2022.
- [347] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.
- [348] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.

- [349] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for bayes-adaptive deep rl via meta-learning. *International Conference of Learning Representations*, 2020.

APPENDIX

Appendix A

ViSaRL

A.1 Environment and Task Details

Meta-World: The observation space is a $64 \times 64 \times 3$ image. The action space $\mathcal{A} \subset \mathbb{R}^4$, is the $\Delta(xyz)$ of the end-effector, and a continuous scalar value for gripper torque. Object and goal positions are randomized at the start of every episode to prevent exploiting spurious correlation or memorizing trajectories to solve the task.

DMControl Generalization Benchmark is built upon the DeepMind Control Suite, a widely used set of continuous control tasks designed to evaluate reinforcement learning algorithms. The observation space is $84 \times 84 \times 3$ RGB images. The action space is continuous and varies depending on the specific task, involving actions such as joint torques or target velocities for controlling the movement of robotic agents.

A key aspect of this benchmark is its emphasis on visual generalization. The DMControl Generalization Benchmark changing the background of the environment to either a static image or a video background. This setup challenges the agent’s ability to generalize, as it must learn image representations that are robust to visual distractions or changes in the background scene.

The benchmark includes a range of tasks such as Walker Walk, Cartpole Swing, Ball Catch, and Finger Spin. Walker Walk requires the agent to control a bipedal walker, coordinating its limbs to achieve stable locomotion and maintain balance while moving forward. In

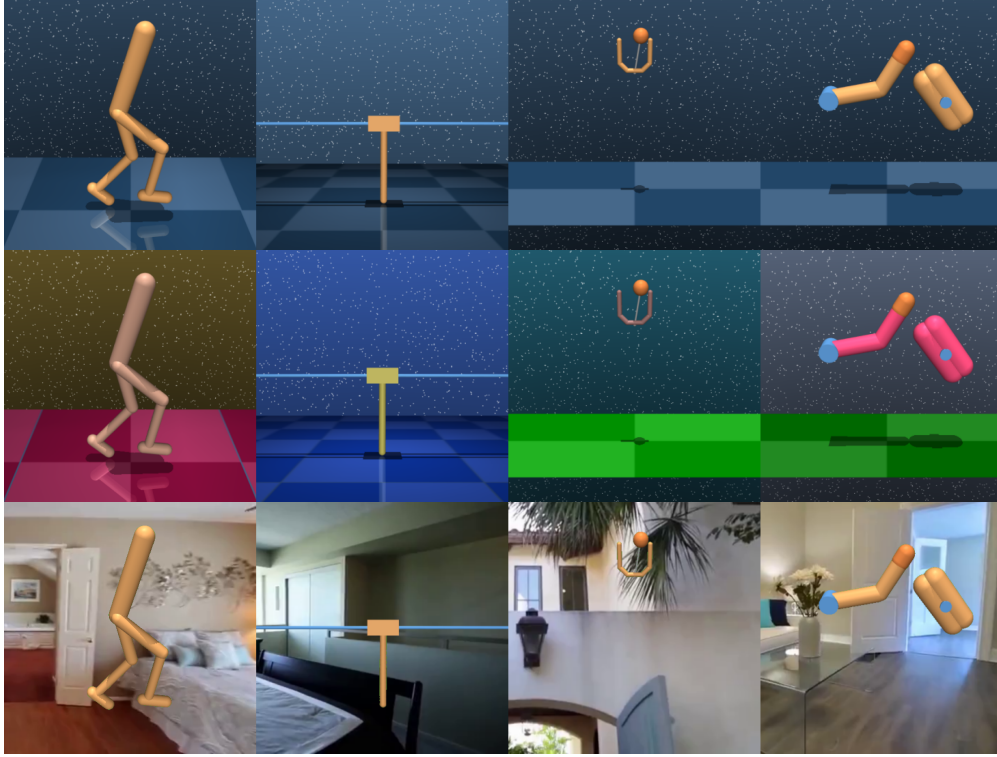


Figure A.1: DMControl Generalization Benchmark (Top) Four continuous control tasks from the DMControl suite. (Middle and Bottom) We evaluate on the color_easy and video_easy settings to test the generalizability of the learned representations.

Cartpole Swing, the agent must swing up and stabilize a pole attached to a cart, balancing it in an upright position. The Ball Catch task requires the agent to manipulate a robotic arm to successfully catch a falling ball. Lastly, Finger Spin involves spinning a small object around its axis using a dexterous robotic finger.

A.2 Hyperparameters

We report the hyperparameters used for Soft Actor-Critic (SAC) training in Table A.1. These settings were used across all DMControl and Meta-World experiments unless otherwise noted.

Table A.1: Hyperparameters used in SAC training.

Hyperparameter	Value
Action repeat	1
Discount factor γ	0.99
Episode length	500
Replay buffer size	1M
Policy learning rate	Adam(lr=1e-4, $\beta_1 = 0.9$, $\beta_2 = 0.999$)
α learning rate	Adam(lr=3e-4, $\beta_1 = 0.5$, $\beta_2 = 0.999$)
Batch size	128
Target network update	2
Target network momentum τ	0.05
Environment steps	1M

A.3 Implementation Details

Implementation details. Table A.1 summarize the hyperparameters in ViSaRL and other baselines. ViSaRL is agnostic to the choice of downstream RL algorithm. For the CNN encoder experiments, we follow the implementation from [27, 171] and train both the encoder and policy end-to-end using Soft-Actor Critic. The encoder consist of a stack of 11 convolutional layers, each with 32 filters of 3×3 kernels, no padding, stride of 2 for the first and 1 for all others. This results in a feature map of dimension $32 \times 12 \times 12$ from an input image of shape $64 \times 64 \times 3$.

The policy head π_θ and action-value functions Q_{ϕ_i} are parameterized by multi-layer perceptrons. The policy head is composed of a linear projection of dimension 100 with normalization followed by 3 linear layers with 1024 hidden units each and a final linear output layer for the action prediction. As the action spaces of DMControl suite and Meta-World are continuous, the policy outputs the mean and variance of a Gaussian distribution over actions.

A.4 Real Robot Imitation Learning

Real Robot Imitation Learning. The downstream policy is trained using standard imitation learning. We train the saliency predictor and Transformer encoder using the same procedure

outlined for the simulation experiments. Given our pretrained visual encoder, we form the state representation as a concatenation of the visual embedding and the robot proprioceptive information (e.g. joint positions). This yields a 271-dimensional state representation.

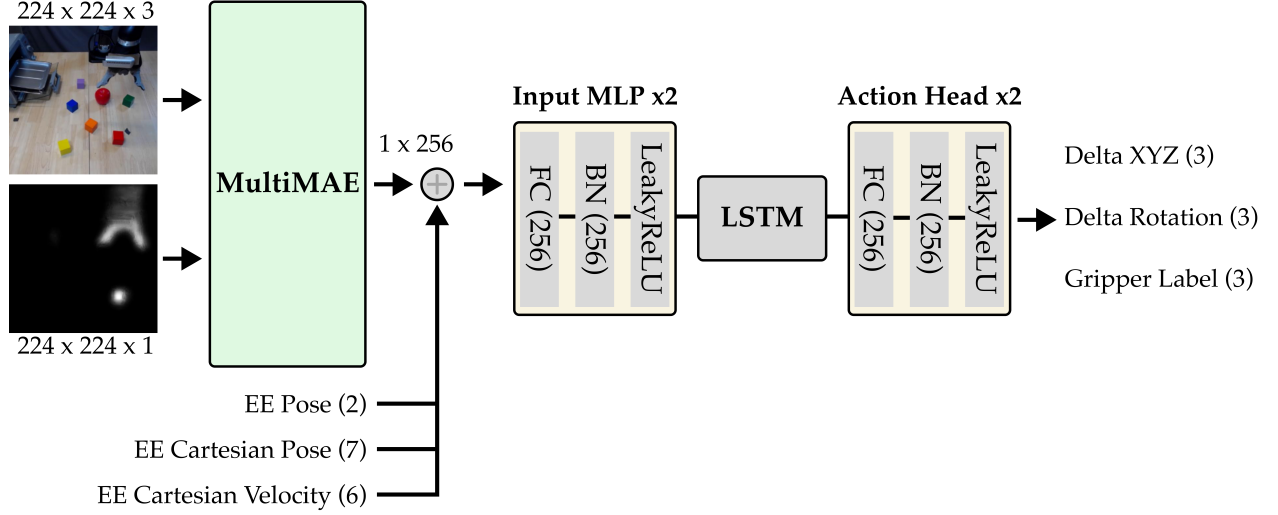


Figure A.2: Robot Policy Architecture We use an LSTM policy with MLPs to embed the input and generate continuous actions. The input is the cropped RGB image from an external camera and the predicted saliency map. The visual input is concatenated with the proprioceptive information.

Table A.2: Hyperparameters used in the real robot experiment.

Hyperparameter	Value
Training epochs	500
Subseq length (H)	15
Batch size	64
Control frequency	5
Gripper weights	[0.585, 0.08, 0.335]
MLP layers	2
MLP hidden sizes	[256, 256]
Normalization	Batch
LSTM hidden size	256

The policy is implemented as an LSTM with 256-dimensional hidden states which autoregressively predicts the actions for the next H timesteps where H is a fixed window. The state is processed by a 2-layer input MLP with hidden sizes [256, 256] followed by a BatchNorm before

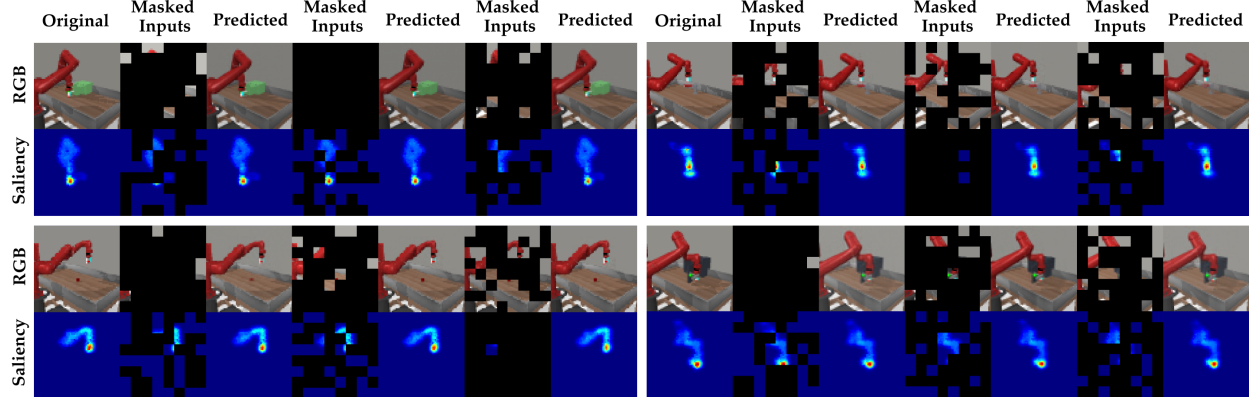


Figure A.3: MultiMAE predictions for different random masks. We visualize the masked predictions for RGB observation from each of the four tasks. For each input image, we randomly sample three different masks from a uniform distribution between RGB and saliency. Only 1/4 of the total patches are unmasked. Even when there are a few unmasked patches from one modality, the reconstructions are still very accurate due to cross-modal interaction. Saliency maps are shown with color for the purposes of visualization.

being inputted to the LSTM. The final hidden state of the LSTM is processed by a 2-layer MLP to predict a continuous action. The full model architecture is shown in Figure A.2. The action space, $\mathcal{A} \in \mathbb{R}^7$, consists of $\Delta(x, y, z, \phi, \theta, \psi)$, and a continuous scalar value for gripper speed.

A.5 Saliency Predictor

Saliency predictor training details. Since we are only using a small dataset for training the saliency predictor, we apply data augmentation to prevent overfitting and improve robustness of the model. After resizing the image to 224×224 , we apply VerticalFlip and HorizontalFlip with a 50% probability and ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.2). We use AdamW optimizer with initial learning rate of $3e-4$ and weight decay 0.005. We reduce the learning rate by 0.1 when loss does not decrease 0.001 with patience 100 and a minimum learning rate of $1e-6$.

We opted to remove one local attention and one global attention decoder layer from the vanilla PiCANet, reducing our inference time per frame from 0.1 seconds to 0.01 seconds. Qualitatively, we observed little degradation in saliency prediction without these layers.

Choice of saliency predictor architecture. Kummerer et al. [165] introduces one of the current state-of-the-art saliency predictors: DeepGaze III. However, DeepGaze III is a scanpath

model. A scanpath model predicts where a participant is likely to fixate given their fixation history. Since scanpath models leverage temporal information from human eye movement, it necessitates that the data collected be from an eye tracker. As our data is collected from clicks, it does not contain any temporal structure and as a result we cannot benefit from scanpath models.

For future works, we plan to explore video saliency and how to exploit temporal patterns in human gaze to further improve the performance of our method. Kummerer et al. [165] also notes that scanpath models have worse performance than static models when the scene contains multiple small salient objects which is typically the case for tabletop manipulation.

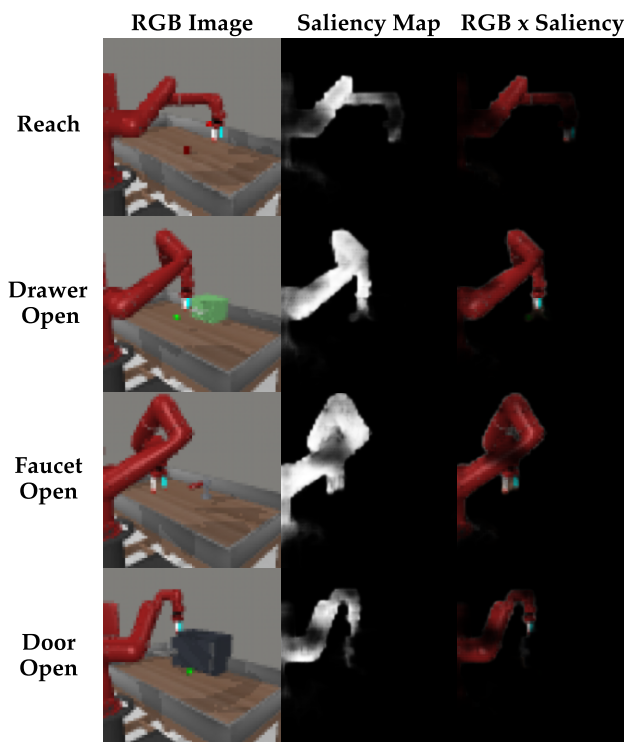


Figure A.4: Zero-shot inference with pretrained PiCANet model for Meta-World observations. We observe that using only a pretrained saliency predictor will only identify the most prominent entity in the scene which in our case is the robot arm. However, task-relevant objects such as the block or the drawer, which are typically smaller in the image frame relative to the robot arm, are not highlighted.

We provide additional quantitative results in Table A.3 comparing the performance of different state-of-the-art saliency predictors including PiCANet [199]. We find PiCANet outperforms the two most recent DeepGaze models in both standard evaluation metrics.

Table A.3: Evaluation metrics measuring the prediction quality different state-of-the-art predictors.

Method	Mean Absolute Error	F_{ζ} score
DeepGaze II [164]	0.0273 ± 0.006	0.7142 ± 0.021
DeepGaze IIE [196]	0.0153 ± 0.006	0.7283 ± 0.024
PiCANet [199]	0.0032 ± 0.002	0.7970 ± 0.015

Zero-shot saliency inference with pretrained PiCANet. We experiment with zero-shot evaluation of a PiCANet saliency model pretrained on the training split of the DUTS dataset [294]. This is currently one of the largest salient object detection datasets with 10,553 training images.

We observe that in a zero-shot setting, the pretrained model consistently segments out the robot arm (see Figure A.4) even in different configurations, which is the most salient object in the scene. However, it fails to identify the goal location or the drawer handle which are critical for completing the task. This demonstrates the need for human-annotated saliency which ViSaRL is able to capture.

Table A.4: Effect of training dataset size on saliency prediction

Number of annotations	Mean Absolute Error	F_{ζ} score
5	0.0086	0.7983
10	0.0063	0.7948
20	0.0053	0.8006
30	0.0020	0.8006

Effect of number of annotated examples on saliency prediction. We provide additional results in Table A.4 investigating how the number of human annotations affects the performance and generalization of the learned saliency model. We conduct an ablation over $x \in \{5, 10, 20, 30\}$ annotated examples. We observe that evaluation metrics improve with more training examples. Qualitatively, there is little difference between the predicted saliency maps using a model trained with 5 examples versus one trained with 30 examples. Even under different initial configurations, the model trained with only 5 annotated examples can generalize the saliency prediction to unseen environment states.

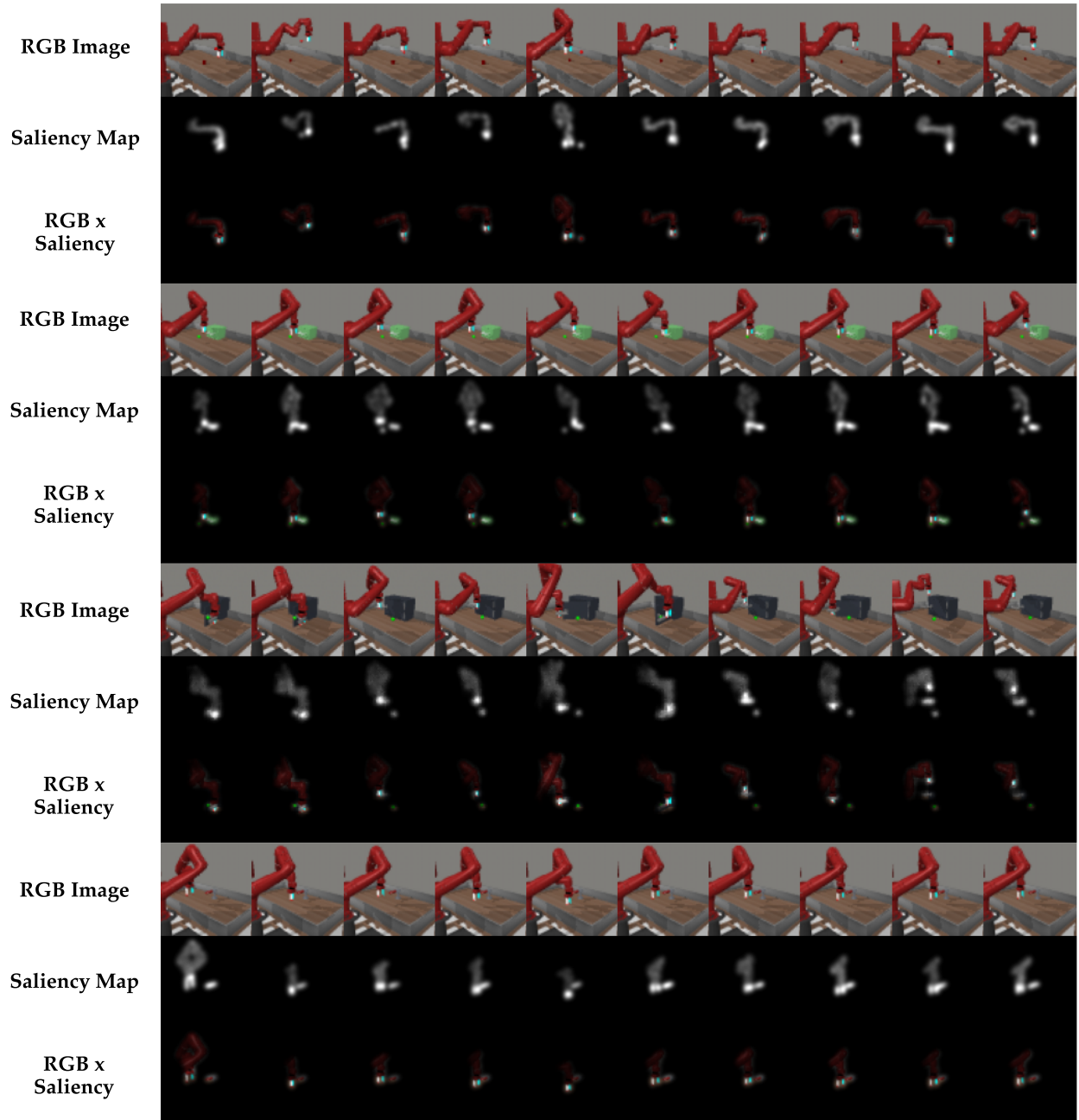


Figure A.5: Visualization of randomly sampled environment observations, their corresponding saliency predicted by a pretrained saliency model, and masked RGB observation from the MetaWorld tasks.

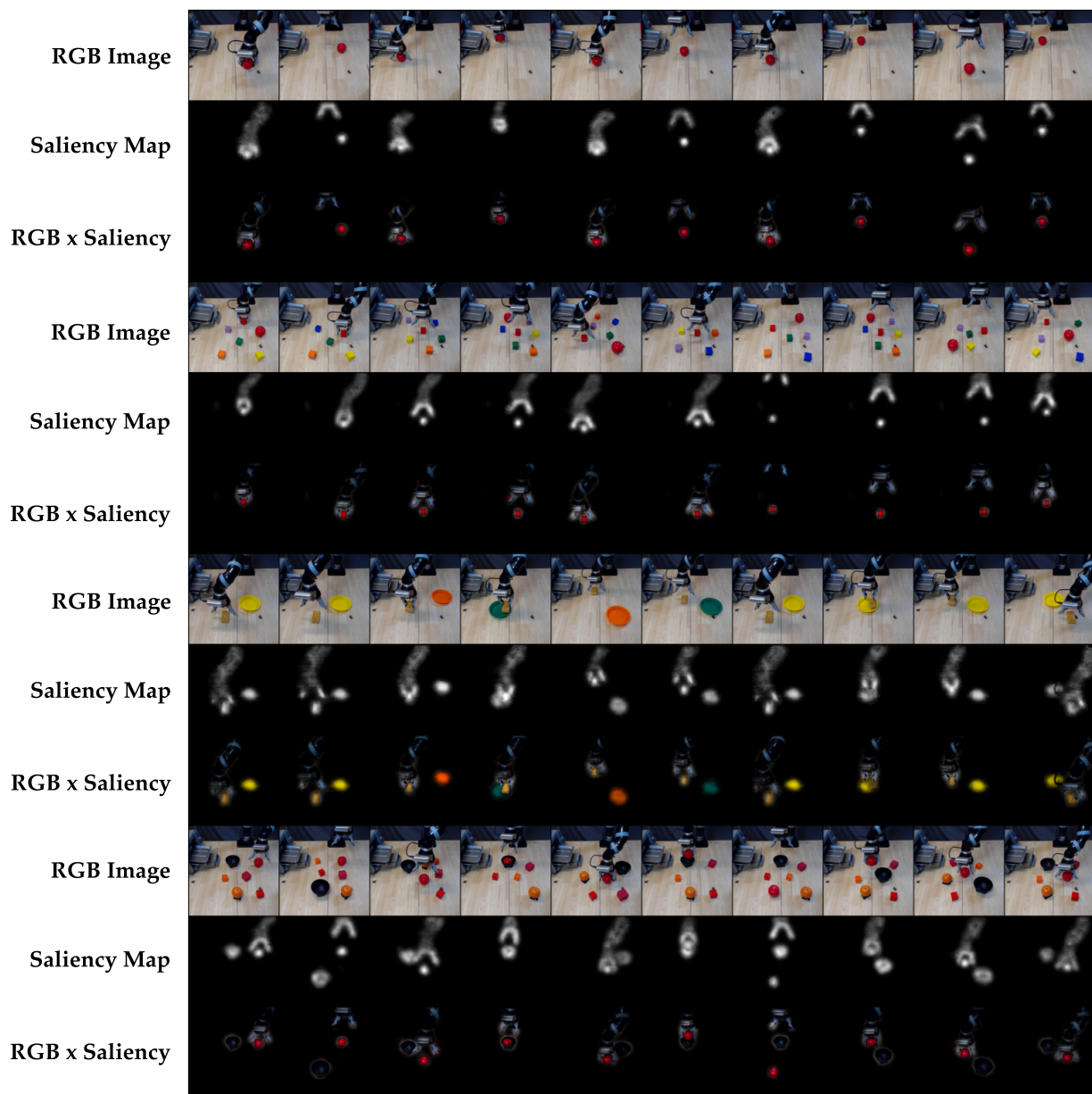
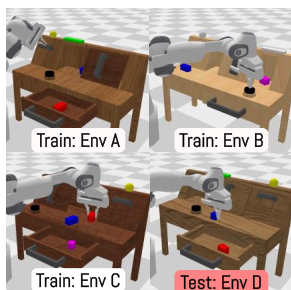


Figure A.6: Visualization of randomly sampled environment observations, their corresponding saliency predicted by a pretrained saliency model, and masked RGB observation from real robot task demonstrations.

Appendix B

HAND

B.1 Environment Details and Hyperparameters



(a) CALVIN [214]



(b) Real-World WidowX-250

Figure B.1: Environments. We retrieve data from a prior dataset to train on *new scenes* in CALVIN. On our real-world WidowX-250 robot, we demonstrate real-world learning from Robometer-retrieved trajectories along with real-time adaptation to long-horizon tasks.

B.1.1 CALVIN.

The CALVIN benchmark is built on top of the PyBullet [63] simulator and involves a 7-DOF Franka Emika Panda Robot arm that manipulates the scene. CALVIN consists of 34 tasks and 4 different environments (ABCD). All environments are equipped with a desk, a sliding door, a drawer, a button that turns on/off an LED, a switch that controls a lightbulb and three different colored blocks (red, blue and pink). These environments differ from each other in the texture of the desk and positions of the objects. CALVIN provides 24 hours of tele-operated unstructured

play data, 35% of which are annotated with language descriptions. We utilize this 35% as a natural way to obtain a smaller subset of the data as the full dataset is very large, but we do not use the task-oriented language instructions. In total, $\mathcal{D}_{\text{play}}$ corresponds to $\sim 17\text{k}$ trajectories for our experiments.

We evaluate on the following tasks:

- **Close Drawer.** For this task, the arm is required to push an opened drawer and close it. The drawer’s degree of openness is randomized.
- **Move Slider Left.** This task requires the robot arm to move a slider located on the desk from the right to the left. The slider position is randomized.
- **Turn On Led.** In this task, the robot arm needs to navigate its way to a button and press down on it such that an LED turns on.
- **Lift Blue Block Table.** For this task, the robot arm needs to pick up a blue block from the table. The location of the blue block on the table is randomized.

B.1.2 Real Robot Experimental Setup



Figure B.2: Real Robot Tasks. We evaluate HAND on 5 different real robot tasks. The last two are long-horizon tasks, requiring more than 100 timesteps of execution.

Hardware Setup. We evaluate HAND on a real-world multi-task kitchen environment using the WidowX robot arm. The WidowX is a 7-DoF robot arm with a two-fingered parallel jaw gripper. Our robot environment setup is shown in [Figure B.1](#). We use an Intel Realsense D435 RGBD camera as a static external camera and a Logitech webcam as an over-the-shoulder camera view. We use a Meta Quest 2 VR headset for teleoperating the robot.

Task-agnostic play dataset. Our play dataset contains a total of 50k transitions collected at 5hz. To encourage diverse behaviors and motions, human teleoperators were instructed to freely

interact with the available objects in the scene without being bound to specific task goals.

Evaluation protocol. The agent is allocated a 100 timestep budget to complete each task. Furthermore, we introduce distractor objects in the scene that are not part of the task so that the policy does not just memorize the expert demonstrations. Moreover, movable task object positions are randomized in a fixed region if applicable. We evaluate on four manipulation tasks described below:

- **Reach Block.** In this task, the robot arm must reach and hover directly above a green block placed on the table. Success is achieved when the gripper remains positioned clearly above the block. Partial success is awarded if the gripper end-effector touches the block without hovering steadily above it.
- **Push Button.** This task requires the robot arm to press the right-side button on a stovetop. Success is achieved upon pressing the button. Partial success is awarded if the robot arm approaches sufficiently close to the button without making contact.
- **Close Microwave.** This task requires the robot to close a microwave door from various starting angles. Partial success is awarded if the robot pushes the door without completely closing it. A successful closure is confirmed by an audible click sound.
- **Put K-Cup in Coffee Machine.**¹ In this task, the robot needs to first pick up the Keurig cup and then transport it to the coffee machine and insert the cup into the cup holder. This task requires precision low-level control as the Keurig cup is small, making it difficult to grasp reliably. Additionally, the cup holder on the coffee machine is just large enough to fit the Keurig cup, leaving small margin of error during the insertion. The coffee machine is fixed to the kitchen stovetop, while the initial location of the Keurig cup is randomized. Given the difficulty of the task, we provide partial success for successfully grasping the Keurig cup.
- **Blend Carrot.** The robot first picks up a toy carrot and then drops it into the blender.

¹https://www.samsclub.com/p/members-mark-gourmet-kitchen-appliances-playset/P990340349?xid=plp_product_2

Once the carrot is inside the blender, it will press a button at the blender base to activate the blender and hold the button for 2 seconds. The location of the blender is static, but the carrot is randomized. Partial success is provided for picking up the carrot and also successfully dropping it into the blender.

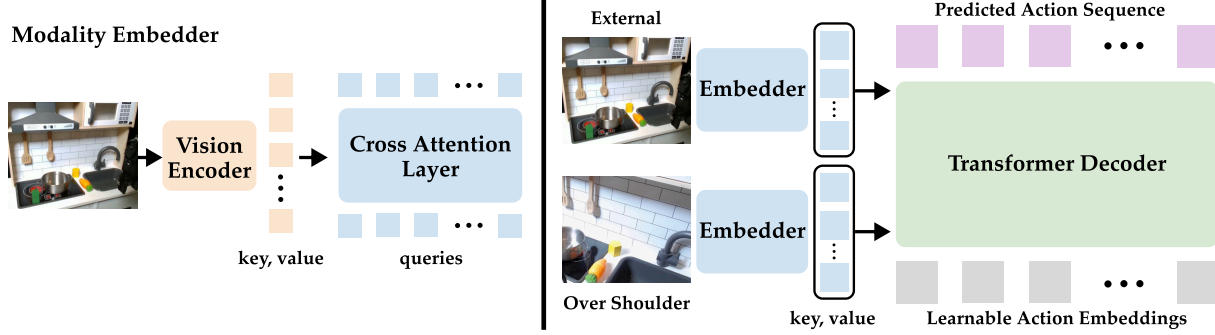


Figure B.3: Real Robot Policy Architecture. (Left) Learnable image embeddings following [295]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [338]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.

Robot Policy. For our policy, we are inspired by the architectural components introduced in Wang et al. [295] and Zhao et al. [338]. A diagram of our policy architecture is shown in Figure B.3. For both external and over-the-shoulder RGB images, we use a pretrained ResNet to first extract 7×7 feature maps and flatten these features across the spatial dimension to create a sequence of d_v dimension tokens where d_v is the output dimension of ResNet. In particular, we use ResNet18 where $d_v = 512$. We feed as input to a causal transformer decoder a sequence learnable action tokens with dimension d . We use the flattened image feature map as the keys and values and apply a cross-attention between the image features and learnable tokens. We concatenate all modality tokens and add additional modality-specific embeddings and sinusoidal positional embeddings.

The policy base is a transformer decoder similar to the one used in ACT [338]. The input sequence to the transformer is a fixed position embedding, with dimensions $k \times 512$ where k is the chunk size and the keys and values are the combined image tokens from the stem. Given the current observation, we predict a chunk of 5 actions, which corresponds to 1 second of execution.

During inference time, we also apply temporal ensembling similar to [338] with $m = 0.5$, which controls the weight of previous actions.

We train the policy for 20k update steps with batch size of 256 and a learning rate of $3e^{-4}$ (around 2 hours of wall time). For behavior cloning policies, the action dimension is 7 comprising of the robot joint pose and the gripper state.

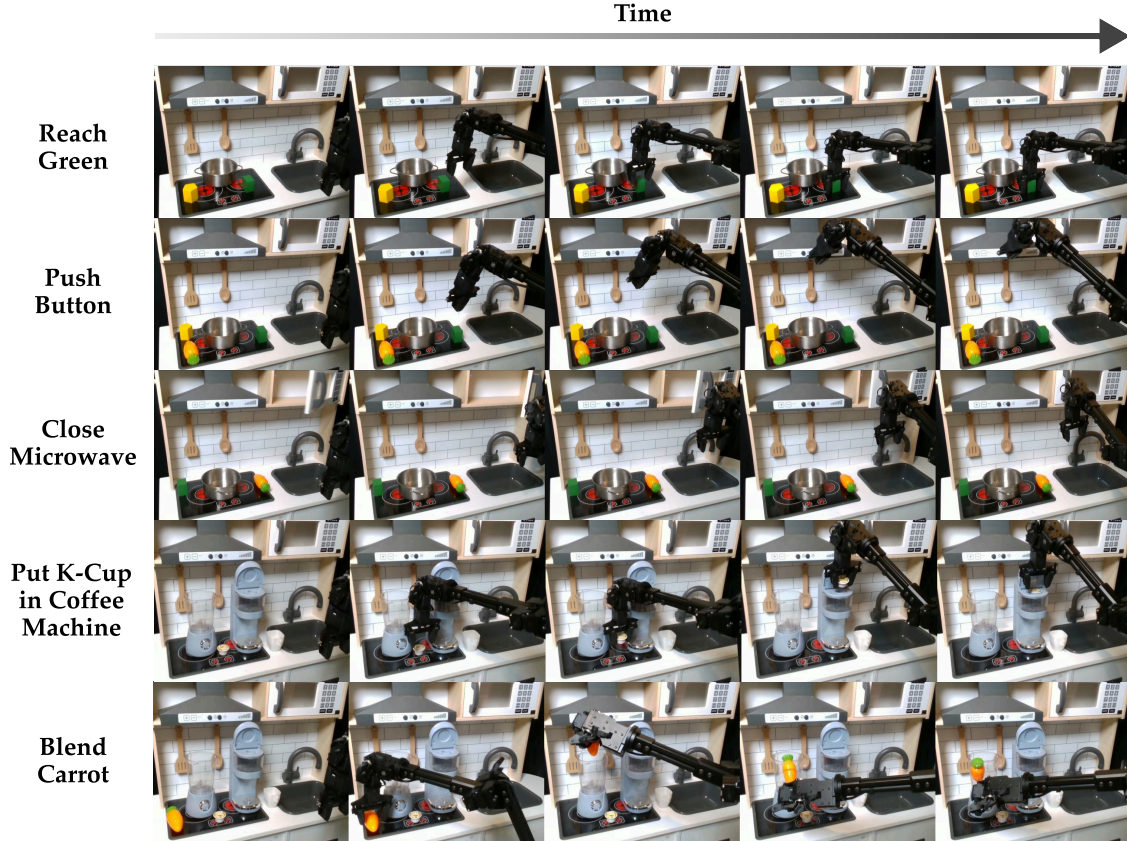


Figure B.4: Task Rollouts

B.2 HAND Algorithm

Algorithm 9 HAND FULL ALGORITHM

Require: Hand demonstrations $\mathcal{D}_{\text{hand}}$, offline play dataset $\mathcal{D}_{\text{play}}$, CoTracker3, Molmo-7B, # retrieved sub-trajectories K , threshold ϵ , DINO, # visual filtered sub-trajectories M

```

/* Policy Pretraining */
1: Train  $\pi_{\text{base}}$  on  $\mathcal{D}_{\text{play}}$  using regular behavior cloning loss  $\mathcal{L}_{BC}$ 
/* Sub-Trajectory Pre-processing */
2:  $\mathcal{T}_{\text{hand}} \leftarrow \text{SubTrajSegmentation}(\mathcal{D}_{\text{hand}}, \epsilon)$   $\triangleright$  Heuristic demo segmentation
3:  $\mathcal{T}_{\text{play}} \leftarrow \text{SubTrajSegmentation}(\mathcal{D}_{\text{play}}, \epsilon)$   $\triangleright$  Heuristic demo segmentation
/* Retrieval using S-DTW and 2D Hand Paths Section 4.3.3 */
4:  $\mathcal{D}_{\text{retrieved}} \leftarrow \{\}$ 
5: for  $\tau_{\text{hand}} \in \mathcal{T}_{\text{play}}$  do
6:    $o_{1:H}^{\text{hand}} \leftarrow \text{image obs sequence of } \tau_{\text{hand}}$ 
7:   for  $\tau_{\text{play}} \in \mathcal{T}_{\text{play}}$  do
8:      $o_{1:T}^{\text{play}} \leftarrow \text{image obs sequence of } \tau_{\text{play}}$ 
9:     /* Visual Filtering */
10:    Compute  $C_{\text{visual}}(o_{1:H}^{\text{hand}}, o_{1:T}^{\text{play}})$  with DINO  $\triangleright$  Equation (4.3.1)
11:     $\mathcal{T}_{\text{play}}^M \leftarrow M$  sub-trajectories with lowest  $C_{\text{visual}}$ 
12:    for  $\tau_{\text{play}} \in \mathcal{T}_{\text{play}}^M$  do
13:       $o_{1:T}^{\text{play}} \leftarrow \text{image obs sequence of } \tau_{\text{play}}$ 
14:       $(x, y)_{\text{hand}} = \text{Molmo}(o_{H/2})$ ,  $(x, y)_{\text{play}} = \text{Molmo}(o_{T/2})$   $\triangleright$  Get middle frame query point
15:       $p_{\text{hand}} = \{(x_t, y_t)_{\text{hand}}\}_1^H = \text{CoTracker3}((x, y)_{\text{hand}})$   $\triangleright$  Track hand point
16:       $p_{\text{play}} = \{(x_t, y_t)_{\text{play}}\}_1^T = \text{CoTracker3}((x, y)_{\text{play}})$   $\triangleright$  Track robot gripper point
17:       $p_{\text{hand}} = p_{\text{hand}}[: -1] - p_{\text{hand}}[1 :]$   $\triangleright$  Convert  $p_{\text{hand}}$  and  $p_{\text{play}}$  to relative 2D paths
18:       $p_{\text{play}} = p_{\text{play}}[: -1] - p_{\text{play}}[1 :]$ 
19:       $(C_{\text{path}}, \tau_{i:j}^{\text{play}}) \leftarrow \text{S-DTW}(p_{\text{hand}}, p_{\text{play}})$   $\triangleright$  Path cost and corresponding retrieved
sequence
20:    Add  $K$  lowest  $C_{\text{path}}$   $\tau_{i:j}^{\text{play}}$  sub-trajectories to  $\mathcal{D}_{\text{retrieved}}$ 
/* Parameter-efficient policy fine-tuning */
21: Insert task-specific adapter LoRA layers  $\theta$  in  $\pi_{\text{base}}$ 
22: Update  $\pi_{\text{base}}$  on  $\mathcal{D}_{\text{retrieved}}$  with loss  $\mathcal{L}_{BC;\theta}$   $\triangleright$  Equation (4.3.2)
23: return  $\pi_{\theta}$ 

```

B.3 User Studies

B.3.1 Efficiency of Hand Demonstrations



Figure B.5: Efficient Demonstrations. Two users, unfamiliar with **HAND** are asked to collect trajectories either via teleoperation (Left) or using their hands (Right). **HAND** retrieval achieves a 50% success rate with the same amount of demonstrations using 3× less time. **STRAP** retrieval is unable to reach 50% even when provided with more expert demonstrations.

In our first study, two users collect 10 demonstrations each either by manually teleoperating using a VR controller or by providing a hand demonstration. For manual teleoperation, we explain to the users how to operate the robot using the VR controller and allow them a couple trials to get accustomed to the interface. For hand demonstrations, we ask the users to mimic the trajectory of the robot end effector using their hands. [Figure B.5](#) shows an example of a user performing both forms of demonstrations. We observe that providing hand demonstrations is significantly more time efficient (over 3×) compared to manual teleoperation. Furthermore, with just a single hand demonstration, we are able to learn a performant policy with 50% success rate, while **STRAP** struggles even when provided 5 expert demonstrations.

B.3.2 Fast Adaptation to Long-Horizon Tasks

We conduct a small study demonstrating that **HAND** enables real-time fast, adaptation to unseen downstream tasks. Snapshots at various stages of this experiment is shown in [Figure B.6](#). In our study, we measure the total time required for a user to provide a hand demonstration of a new target task to evaluating the performance of a fine-tuned policy. The hand demonstration is simple



Figure B.6: Fast Adaptation. We conduct a small-scale user study to demonstrate HAND’s ability to learn robot policies in real-time. From providing the hand demonstration (Left), to retrieval and fine-tuning a base policy (Middle), to evaluating the policy (Right), we show that **HAND** can learn to solve the Blend Carrot task with over 70% success rate in less than 3 minutes.

to provide and typically takes between 10 – 15 seconds to collect. Data preprocessing, which involves computing the 2D path features of the hand demonstration and performing retrieval, takes around 30 – 40 seconds. We assume that the offline play dataset is already preprocessed prior to the study and we do not include this time in our estimate. We also assume a base policy has already been trained on this data; however, it performs poorly on the target task. We fine-tune the base policy with 4 LoRA adapter layers for 1000 batch updates, which takes ~ 2 minutes on a NVIDIA 4070 GPU. The resulting policy, which took less than 3 minutes to train and achieves over 70% success rate, highlighting the efficacy of HAND for real-time policy learning. An uncut video of this study can be found on our project website at <https://handretrieval.github.io/>.

B.4 Qualitative Retrieval Analysis

In [Figure B.7](#), we provide more qualitative results comparing STRAP retrieval results to HAND on each of our real robot tasks. Across all tasks, HAND retrieves more relevant trajectories that perform the task demonstrated by the human hand.

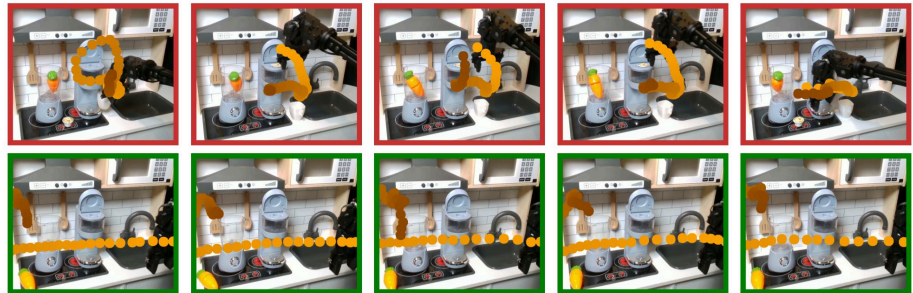
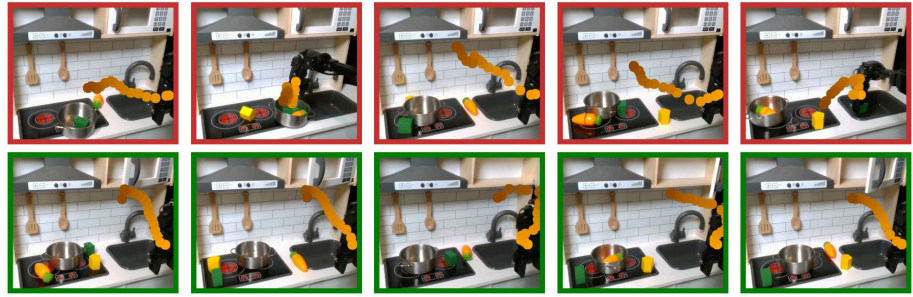
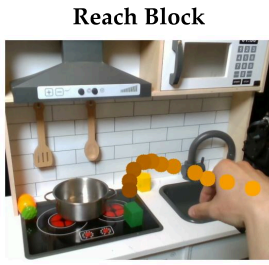


Figure B.7: Qualitative Retrieval Examples. We show the top 5 matches from D_{play} for **STRAP** (top) and **HAND** (bottom) provided the hand demonstration for each of our evaluation tasks.

B.5 CALVIN Results

Table B.1: CALVIN Close Drawer: Performance with and without expert demonstrations

Method	K=25	K=50	K=100	K=250
<i>With Expert</i>				
FT	0.425 $\pm$ 0.059	-	-	-
Flow	0.694 $\pm$ 0.089	0.797 $\pm$ 0.045	0.633 $\pm$ 0.127	0.747 $\pm$ 0.039
STRAP	0.481 $\pm$ 0.119	0.286 $\pm$ 0.073	0.703 $\pm$ 0.075	0.600 $\pm$ 0.085
<i>Without Expert</i>				
π_{base}	0.233 $\pm$ 0.024	-	-	-
CLIP	0.003 $\pm$ 0.004	0.000 $\pm$ 0.000	0.000 $\pm$ 0.000	0.000 $\pm$ 0.000
Flow	0.808 $\pm$ 0.080	0.831 $\pm$ 0.058	0.533 $\pm$ 0.106	0.653 $\pm$ 0.055
STRAP	0.000 $\pm$ 0.000	0.011 $\pm$ 0.010	0.006 $\pm$ 0.008	0.031 $\pm$ 0.004
HAND(+3D, -VF, -CW)	0.994 $\pm$ 0.004	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000
HAND(-VF, -CW)	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000
HAND(-VF)	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	0.997 $\pm$ 0.004	1.000 $\pm$ 0.000
HAND	0.828 $\pm$ 0.169	0.464 $\pm$ 0.061	0.536 $\pm$ 0.082	0.436 $\pm$ 0.136

Table B.2: CALVIN Move Slider Left: Performance with and without expert demonstrations

Method	K=25	K=50	K=100	K=250
<i>With Expert</i>				
FT	0.564 $\pm$ 0.309	-	-	-
Flow	0.092 $\pm$ 0.038	0.086 $\pm$ 0.017	0.156 $\pm$ 0.046	0.039 $\pm$ 0.039
STRAP	0.053 $\pm$ 0.034	0.075 $\pm$ 0.012	0.111 $\pm$ 0.014	0.094 $\pm$ 0.037
<i>Without Expert</i>				
π_{base}	0.011 $\pm$ 0.010	-	-	-
CLIP	0.017 $\pm$ 0.024	0.033 $\pm$ 0.047	0.006 $\pm$ 0.004	0.031 $\pm$ 0.024
Flow	0.000 $\pm$ 0.000	0.247 $\pm$ 0.116	0.094 $\pm$ 0.046	0.053 $\pm$ 0.014
STRAP	0.058 $\pm$ 0.018	0.122 $\pm$ 0.022	0.075 $\pm$ 0.025	0.028 $\pm$ 0.024
HAND(+3D, -VF, -CW)	0.028 $\pm$ 0.008	0.047 $\pm$ 0.010	0.192 $\pm$ 0.049	0.139 $\pm$ 0.040
HAND(-VF, -CW)	0.186 $\pm$ 0.088	0.081 $\pm$ 0.017	0.364 $\pm$ 0.149	0.619 $\pm$ 0.092
HAND(-VF)	0.069 $\pm$ 0.042	0.167 $\pm$ 0.056	0.200 $\pm$ 0.123	0.325 $\pm$ 0.014
HAND	0.647 $\pm$ 0.229	0.483 $\pm$ 0.041	0.636 $\pm$ 0.103	0.431 $\pm$ 0.107

Table B.3: CALVIN Turn On LED: Performance with and without expert demonstrations

Method	K=25	K=50	K=100	K=250
<i>With Expert</i>				
FT	0.000 $\pm$ 0.000	-	-	-
Flow	0.131 $\pm$ 0.085	0.344 $\pm$ 0.092	0.697 $\pm$ 0.082	0.581 $\pm$ 0.134
STRAP	0.200 $\pm$ 0.147	0.125 $\pm$ 0.042	0.056 $\pm$ 0.017	0.372 $\pm$ 0.220
<i>Without Expert</i>				
π_{base}	0.036 $\pm$ 0.014	-	-	-
CLIP	0.025 $\pm$ 0.035	0.006 $\pm$ 0.008	0.019 $\pm$ 0.016	0.000 $\pm$ 0.000
Flow	0.017 $\pm$ 0.024	0.011 $\pm$ 0.008	0.364 $\pm$ 0.147	0.436 $\pm$ 0.031
STRAP	0.500 $\pm$ 0.131	0.600 $\pm$ 0.184	0.525 $\pm$ 0.150	0.633 $\pm$ 0.112
HAND(+3D, -VF, -CW)	0.333 $\pm$ 0.111	0.661 $\pm$ 0.093	0.814 $\pm$ 0.059	0.489 $\pm$ 0.136
HAND(-VF, -CW)	0.675 $\pm$ 0.065	0.719 $\pm$ 0.155	0.886 $\pm$ 0.032	0.431 $\pm$ 0.103
HAND(-VF)	0.428 $\pm$ 0.016	0.467 $\pm$ 0.138	0.828 $\pm$ 0.058	0.881 $\pm$ 0.034
HAND	0.136 $\pm$ 0.102	0.278 $\pm$ 0.073	0.186 $\pm$ 0.051	0.094 $\pm$ 0.017

B.6 Real Robot Results

Table B.4: Real-world expert demonstrations ($N = 3$). Success rates out of 10 trials per task.

Task \ Method	π_{base}	Flow	STRAP				HAND (-VF)	HAND		
		$K=25$	$K=10$	$K=25$	$K=50$		$K=25$	$K=10$	$K=25$	$K=50$
Reach Green Block	1.0	1.5	2.5	2.0	2.5		2.5	6.0	7.5	5.0
Press Button	0.0	0.0	5.5	5.0	2.5		4.0	8.5	5.0	4.0
Close Microwave	0.5	0.5	5.0	2.5	4.0		3.0	7.0	8.0	4.5

Table B.5: Real-world hand demonstration ($N = 1$). Success rates out of 10 trials per task.

Task \ Method	π_{base}	Flow	STRAP				HAND (-VF)	HAND		
		$K=25$	$K=10$	$K=25$	$K=50$		$K=25$	$K=10$	$K=25$	$K=50$
Reach Green Block	1.0	0.0	3.0	1.0	1.0		1.0	6.5	7.0	6.0
Press Button	0.0	0.5	1.5	0.0	0.5		0.0	4.5	6.0	3.5
Close Microwave	0.5	0.0	0.0	0.0	0.0		5.0	8.0	4.0	1.0

Appendix C

CLAM

C.1 Additional Experimental Results

This section reports additional experimental results for CLAM, covering state-based MetaWorld performance, ablations of design choices, and additional transfer experiments on CALVIN.

C.1.1 State-based Results MetaWorld

We present additional state-based results for the MetaWorld tasks. We experiment with using both 50 and 100 trajectories for $\mathcal{D}_{\text{labeled}}$. In both data regimes, CLAM significantly outperforms the baseline across all tasks by +12% and +35% task success rate to the next best baseline respectively. We find that in more fine-grained manipulation tasks like Shelf Place, CLAM struggles to accurately pick up the block with lower amounts of action-labeled data. However, as we increase the amount of labeled data and hence coverage in the state-action space for training our action decoder, the performance improves dramatically, from 28% to 93% task success with **TF-CLAM**.

Table C.1: State-Based Input Results (50 action-labeled trajectories)

	Assembly	Bin Picking	Peg Insert Side	Shelf Place	Average
BC-AL	0.19 $\pm$ 0.03	0.21 $\pm$ 0.12	0.23 $\pm$ 0.09	0.00 $\pm$ 0.00	0.16
LAP0	0.05 $\pm$ 0.06	0.00 $\pm$ 0.00	0.06 $\pm$ 0.03	0.02 $\pm$ 0.02	0.03
LAPA	0.07 $\pm$ 0.04	0.02 $\pm$ 0.01	0.09 $\pm$ 0.01	0.04 $\pm$ 0.02	0.05
DynaMo	0.08 $\pm$ 0.02	0.04 $\pm$ 0.02	0.08 $\pm$ 0.04	0.06 $\pm$ 0.02	0.06
VPT	0.24 $\pm$ 0.06	0.02 $\pm$ 0.02	0.34 $\pm$ 0.06	0.00 $\pm$ 0.00	0.15
MLP-CLAM*	0.68 $\pm$ 0.03	0.54 $\pm$ 0.08	0.44 $\pm$ 0.03	0.26 $\pm$ 0.04	0.48
TF-CLAM*	0.81 $\pm$ 0.03	0.74 $\pm$ 0.04	0.55 $\pm$ 0.12	0.28 $\pm$ 0.02	0.60
BC-Expert	0.84 $\pm$ 0.03	0.76 $\pm$ 0.03	0.80 $\pm$ 0.07	0.93 $\pm$ 0.05	0.83

Table C.2: State-Based Input Results (100 action-labeled trajectories). We report average task success rate for MetaWorld tasks. Maroon highlights the best method in that environment except for BC-Expert which is trained with expert, labeled data. Our method TF-CLAM, which uses a transformer IDM/FDM, outperforms the baselines on all tasks. *Methods with an asterisk are ours.

	Assembly	Bin Picking	Peg Insert Side	Shelf Place	Average
BC-AL	0.34 $\pm$ 0.05	0.27 $\pm$ 0.12	0.29 $\pm$ 0.07	0.00 $\pm$ 0.00	0.24
LAP0	0.15 $\pm$ 0.04	0.02 $\pm$ 0.03	0.17 $\pm$ 0.04	0.06 $\pm$ 0.08	0.13
LAPA	0.24 $\pm$ 0.07	0.15 $\pm$ 0.01	0.25 $\pm$ 0.02	0.12 $\pm$ 0.02	0.21
DynaMo	0.10 $\pm$ 0.03	0.06 $\pm$ 0.03	0.12 $\pm$ 0.04	0.08 $\pm$ 0.02	0.13
VPT	0.40 $\pm$ 0.08	0.05 $\pm$ 0.02	0.49 $\pm$ 0.06	0.02 $\pm$ 0.00	0.28
MLP-CLAM*	0.53 $\pm$ 0.04	0.68 $\pm$ 0.05	0.58 $\pm$ 0.04	0.72 $\pm$ 0.04	0.63
TF-CLAM*	0.91 $\pm$ 0.03	0.82 $\pm$ 0.03	0.79 $\pm$ 0.07	0.93 $\pm$ 0.02	0.83
BC-Expert	1.00 $\pm$ 0.00	0.94 $\pm$ 0.05	0.91 $\pm$ 0.03	0.93 $\pm$ 0.00	0.87

C.1.2 CLAM Ablations

Analysis of the action decoder weight β . We conduct a study where we vary the weight on the action decoder loss during joint training across $\beta \in [0, 0.001, 0.01, 1, 5]$. When $\beta = 0$, this is equivalent to no action decoder training which will result in a random action decoder. Since the action decoder is not trained at all, we do not expect the model to be able to perform the task at all. As we increase the loss coefficient, we observe improved performance as the decoder is now able to ground latent actions to environment actions. We find that an equal weighting between

Table C.3: CLAM ablation study (state-based). We ablate discrete vs. continuous actions and with/without joint training. Both are critical for a performant policy; joint training improves results across tasks.

	Assembly	Bin Picking	Peg Insert	Side Shelf Place
Discrete, No Joint Training	0.15 $\pm$ 0.04	0.13 $\pm$ 0.04	0.16 $\pm$ 0.04	0.15 $\pm$ 0.06
Discrete, Joint Training	0.18 $\pm$ 0.05	0.18 $\pm$ 0.02	0.12 $\pm$ 0.03	0.21 $\pm$ 0.03
Continuous, No Joint Training	0.32 $\pm$ 0.06	0.24 $\pm$ 0.05	0.15 $\pm$ 0.04	0.32 $\pm$ 0.04
Continuous, Joint Training (Ours)	0.91 $\pm$ 0.03	0.82 $\pm$ 0.03	0.79 $\pm$ 0.07	0.93 $\pm$ 0.02

the reconstruction and action decoder training yields the best result, although CLAM is still quite robust to the value of β . Results for different β coefficients on the MetaWorld Assembly task are shown in Table C.4.

Table C.4: Varying action decoder loss weight β . Increasing the decoder weight improves downstream task performance.

Action Decoder Weight (β)	Task Success
0	0.00 $\pm$ 0.00
0.001	0.42 $\pm$ 0.04
0.01	0.52 $\pm$ 0.03
1	0.58 $\pm$ 0.04
5	0.53 $\pm$ 0.03

C.1.3 Additional Transfer Experiments in CALVIN

We present additional experiments to study the transferability of our methods. In particular, we conduct experiments in the CALVIN benchmark [214], which provides a dataset of ~ 24 hours of task-agnostic play data. We use trajectories from 10% of the full dataset (13170 trajectories across 30 different tasks) as

Table C.5: CALVIN generalization results. CLAM generalizes better than VPT on target tasks without labeled demonstrations for those tasks.

	Close Drawer	Slider Left
BC-AL	0.06 $\pm$ 0.01	0.06 $\pm$ 0.01
VPT	0.29 $\pm$ 0.04	0.11 $\pm$ 0.03
TF-CLAM (Ours)	0.33 $\pm$ 0.08	0.26 $\pm$ 0.08

$D_{\text{unlabeled}}$ to pretrain CLAM and sample data

from five different tasks for D_{labeled} .

Notably, we find that CLAM can transfer to target tasks for which it has seen unlabeled but *no action-labeled* data, similar to our real robot experiments. In this study, the target task has only *unlabeled* expert data. Specifically, we evaluate on the Close Drawer and Slide Left tasks. We include the unlabeled dataset for these tasks in the LAM pre-training. However, we do not have access to action-labeled data for these tasks. Instead, we select action-labeled data from other tasks that are likely to span the complete state space.

Table C.5 shows that **BC-AL** fails in this task because the action-labeled training data is from different tasks and thus out-of-distribution. Occasionally, **BC-AL** is able to solve the task by random chance. In the Close Drawer task, **VPT** performs comparably to **Trans.-CLAM**. However, the motions of the robot arm in successful rollouts are much more visually similar to the expert trajectories indicating that **Trans.-CLAM** learns a more meaningful latent action space. For reference, we attach videos of each method’s behavior on our website. In the Slider Left task, which requires more precision, **Trans.-CLAM** achieves more than $2\times$ the success rate of **VPT** without ever seeing the target task in the action-labeled dataset.

C.2 Trivial Solutions

Our method aims to learn an IDM-based labeler that provides latent action labels with two goals: 1) the labels can be used in place of ground truth action labels to learn a behavioral cloning policy and 2) the labels can be mapped to ground truth actions using a learned action decoder. Any labeling that can satisfy these requirements is useful to our method and non-trivial. We do not impose any specific prior structure on the latent labels, as long as they satisfy these requirements. However, we agree that regularizing priors might improve the performance of our method, which is a promising direction for future work.

To verify no trivial shortcuts occur, we compare the reconstruction loss of our FDM output $\hat{o}_{t+1}$ (the predicted next observation) with a trivial baseline FDM that always outputs the pre-

vious observation o_t . We find that our method achieves lower loss than is possible by trivially outputting the previous observation, which indicates that our model does not resort to this shortcut. Table C.6 below shows these results.

Additionally, the action decoder only receives the label z_t that is output by our learned policy. Thus, in order for a policy to achieve a task, the labels need to meaningfully correspond to correct actions. If the LAM solves the reconstruction and action decoding outputs in a trivial way using shortcuts in such a way that the labels don't hold any useful information, the action decoder would have no way of knowing what actions to output that correctly solve the task. The positive results of our method are strong indicators that our IDM/FDM structure has not reached a trivial solution.

Table C.6: Image reconstruction loss vs. cheating loss

Task	Reconstruction loss	Cheat loss
Assembly	0.0004	0.0007
Bin-Picking	0.0005	0.0008
Peg Insert Side	0.0005	0.0008
Shelf Place	0.0003	0.0009

C.3 Detailed Algorithms

Below, we provide a detailed algorithm of training CLAM and latent action policy in Algorithm 10.

We also provide the pseudocode for inference rollouts in Algorithm 4.

Algorithm 11 Inference Time Rollout

```

1: Input: Action Decoder  $p_\omega$ , Latent Policy  $\pi_\theta$ 
2:  $o_t = \text{env.reset}()$ 
3: while not done do
4:    $z_t = \pi_\theta(\cdot \mid o_t)$  ▷ infer latent action
5:    $a_t = p_\omega(\cdot \mid z_t)$  ▷ decode latent action
6:    $o_t, \text{done} = \text{env.step}(a_t)$ 

```

Algorithm 10 CLAM w/ Joint Action Decoder Training

```
1: Input:  $\mathcal{D}_{\text{unlabeled}}, \mathcal{D}_{\text{labeled}}, \mathcal{D}_{\text{unlabeled\_expert}}, \text{IDM } f_{\phi}, \text{FDM } g_{\psi}, \text{Action Decoder } p_{\omega}, \text{Latent Action Policy } p_{\theta}$   
    $N_C$ : number of CLAM update steps  
    $N_P$ : number of policy update steps  
    $K$ : train action decoder every  
   # Stage 1: Train CLAM and Action Decoder  
2: for iter = 1 to  $N_C$  do  
3:   Sample  $(o_t, o_{t+1})$  from  $\mathcal{D}_{\text{unlabelled}}$   
4:    $z_t = f_{\phi}(\cdot \mid o_t, o_{t+1})$  ▷ infer latent action  
5:    $\hat{o}_{t+1} = g_{\psi}(\cdot \mid o_t, z_t)$  ▷ predict next observation  
6:    $\mathcal{L}_{\text{MSE}}(\phi, \psi) = \|o_{t+1} - \hat{o}_{t+1}\|_2^2$  ▷ update IDM and FDM  
   # Jointly Train Action Decoder  
7:   if iter %  $K == 0$  then  
8:     Sample  $\{(o_t, a_t, o_{t+1})\}_{i=1}^B$  from  $\mathcal{D}_{\text{labeled}}$  ▷ get batch for action decoder training  
9:      $z_t = f_{\phi}(\cdot \mid o_t, o_{t+1})$  ▷ infer latent action  
10:     $\hat{a}_t = p_{\omega}(\cdot \mid z_t)$  ▷ decode latent action  
11:     $\mathcal{L}_{\text{action\_decoder}}(\phi, \omega) = \text{MSE}(\hat{a}_t, a_t)$  ▷ update action decoder and IDM  
   # Stage 2: Train Latent Action Policy  
12: for iter = 1 to  $N_P$  do  
13:   Sample  $N_D$  demos from  $\mathcal{D}_{\text{unlabeled\_expert}}, \{\tau_i\}_{i=0}^{N_D}$   
14:    $\tau_i^* = \{o_1, f_{\phi}(o_1, o_2), o_2, \dots, f_{\phi}(o_{T-1}, o_T), o_T\}$  ▷ annotate transitions with latent actions  
15:    $\mathcal{L}_{\text{MSE}}(\theta) = \|\hat{z}_{1:T} - z_{1:T}\|_2^2$  ▷ update latent policy
```

C.4 Hyperparameters

We provide detailed hyperparameters for training baseline methods and CLAM.

Table C.7: BC Hyperparameters

Hyperparameter	Value
Batch Size	64
Num training updates	150,000
MLP Hidden dims	[1024, 1024]
Number of eval rollouts	40
Clip grad norm	1.0
Optimizer	Adam
Learning Rate (LR)	3e-4
Epsilon	1e-5
Weight decay	0.01
Num of warmup updates	25,000
LR Scheduler	ConstantLR

Table C.8: CLAM Pretraining Hyperparameters

Hyperparameter	Value
Num updates	500,000
Train action decoder every	2
Action Decoder batch size	128
Action Decoder loss weight	1
Action Decoder hidden dim	[1024, 1024, 1024]
Action Decoder embedding dim	512
Reconstruction loss weight	1
Latent action dim	16
Context len	2
Embedding dim	128

Table C.9: Transformer CLAM Model Hyperparameters

Hyperparameter	Value
Num encoder layers	3
Num decoder layers	3
Model dimension	256
Feedforward dimension	2048
Num attention head	4
Dropout	0.1
Pre norm	False
Feedforward activation	GeLU
Position Encoding	Learned

Table C.10: CALVIN Transformer CLAM Model Hyperparameters

Hyperparameter	Value
Num encoder layers	6
Num decoder layers	6
Model dimension	512
Feedforward dimension	2048
Num attention head	8
Dropout	0.1
Pre norm	False
Feedforward activation	GeLU
Position Encoding	Learned

Table C.11: MetaWorld Environment Hyperparameters

Hyperparameter	Value
Max episode steps	100
State dim	39
Action dim	4
Image shape	[84, 84, 3]
Num frame stack	3
Action repeat	2

Table C.12: CALVIN Environment Hyperparameters

Hyperparameter	Value
Max episode steps	200
State dim	39
Action dim	7
Image shape	[84, 84, 3]
Num frame stack	1
Action repeat	7

Table C.13: Data splits (# trajectories).

	MuJoCo	MetaWorld	Real Robot
$\mathcal{D}_{\text{unlabeled}}$	1000	1000	~500
$\mathcal{D}_{\text{labeled}}$	50	50	50
$\mathcal{D}_{\text{unlabeled_expert}}$	20	20	30/50

C.5 Environment Details

We benchmark our method on DMControl [284], Meta-World [325], and CALVIN [214] without modification. All domains are continuous control environments and we use a fixed episode length and no termination conditions. Figure C.1 provides illustrations of each task we evaluate on.

For DMControl tasks, we report normalized return following [91] which normalizes the scores for each environment against an expert score between 0 and 1: $\text{normalized score} = \frac{(\text{score} - \text{random score})}{(\text{expert score} - \text{random score})}$. In MetaWorld and CALVIN, we evaluate based on task success rate and we consider an episode successful if the task is completed at *any* step in a given episode.

MuJoCo Locomotion. We evaluate on two continuous control tasks from the MuJoCo locomotion benchmark [284]: Hopper and HalfCheetah. Hopper is a single-legged robot, and HalfCheetah is a planar bipedal robot with a torso and two articulated legs. The action space for these tasks represents joint torques applied at the robot’s actuators. The action space is a 3-dimensional and 6-dimensional respectively for Hopper and HalfCheetah. The state space is a

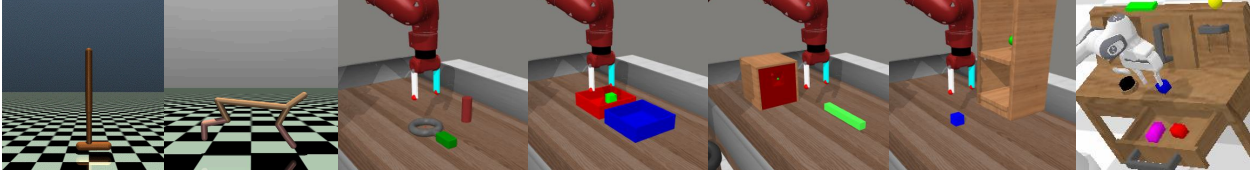


Figure C.1: Evaluation environments in simulation. We evaluate our approach on both *locomotion* tasks from the DMControl benchmark (Hopper and HalfCheetah) and *manipulation* tasks (Assembly, Bin Picking, Peg Insert Side, and Shelf Place) from the MetaWorld benchmark. We also evaluate in CALVIN with the Close Drawer and Slider Left tasks.

11-dimensional vector and 17-dimensional vector respectively comprising joint angles, joint velocities, and the global position of the robot’s torso. The goal in both tasks is to maximize forward velocity while minimizing a control cost for taking large actions and maintaining stability. Each episode is 1000 timesteps.

MetaWorld. We evaluate on four continuous control task from the MetaWorld benchmark [325]. The MetaWorld benchmark is designed for multitask and meta-reinforcement learning research. Each task shares the same embodiment (a 6-DOF Sawyer robot arm), observation space, and action space. The action space represents end-effector deltas, $(\Delta x, \Delta y, \Delta z, \beta)$ where β is a binary value (0, 1) for the gripper action. The state space is a 39-dimensional vector which comprises of a framestack of [curr_obs, prev_obs, pos_goal] where curr_obs and prev_obs are both 18-dimensional and pos_goal is a 3-dimensional vector representing the goal position. The observation is a single vector consisting of the end effector position, gripper’s distance apart, and each object’s position and quaternion.

CALVIN is an open-source simulated benchmark to learn long-horizon language-conditioned tasks. For our experiments, we employ scene D consisting of a 7-DOF Franka Emika Panda robot arm with a gripper and a desk with a sliding door and a drawer that can be opened and closed. CALVIN provides 6 hours of teleoperated data for each of the 4 environments for a total of 24 hours of play data, 35% of which contains crowd-sourced language annotations. To maximize the use of the available play data, we employ the provided language annotation tool and divide the dataset with respect to these annotations. For our experiments, we use the 15-dimensional proprioceptive state appended with the 24-dimensional scene observation along with the 7-dimensional

relative cartesian actions.

Table C.14: Task observation and action dimensions.

Task	Observation Dim	Action Dim
Hopper	11	3
HalfCheetah	17	6
Assembly	39	4
Bin Picking	39	4
Shelf Place	39	4
Peg Insert Side	39	4
Close Drawer	39	7
Slider Left	39	7

Table C.15: Environment comparison summary.

	DMControl	Meta-World	CALVIN
Episode Length	1000	200	200
Action Repeat	2	2	8
Effective Length	500	100	200
Performance Metric	Normalized Return	Task Success	Task Success

C.6 Real Robot Experiments

Hardware Setup. We evaluate CLAM on a real-world multi-task kitchen environment using the WidowX robot arm. The WidowX is a 7-DoF robot arm with a two-fingered parallel jaw gripper. Our robot environment setup is shown in Figure 5.3. We use an Intel Realsense D435 RGBD camera as a fixed external camera and a Logitech webcam as an over-the-shoulder camera view. We use a Meta Quest 2 VR headset to teleoperate the robot.

Task-agnostic play dataset. Our play dataset contains a total of 50k transitions collected at 5hz. To encourage diverse behaviors and broad state-action coverage, human teleoperators were instructed to freely interact with the available objects in the scene without being bound

to specific task goals. Note, we do not require that the play trajectories be optimal, but that it provide diverse coverage of the state-action space.

Evaluation protocol. The agent is allocated a 100 timestep budget to complete each task. For each task, we collect a total of 30 expert demonstrations, except for the long-horizon task for which we use 50 expert demonstrations. For training the latent action policy, we remove the collected action labels and replace them with the latent actions learned by our LAM. Furthermore, we introduce distractor objects in the scene that are not part of the task so that the policy does not just memorize the expert demonstrations. Moreover, movable task object positions are randomized in a fixed region if applicable. We evaluate on four manipulation tasks described below:

- **Reach Block.** This task requires the arm to reach to the position of the block and hover above it. The task is considered successful if the robot gripper is above the green block. We provide partial success if the gripper end effector touches the block.
- **Push Button.** This task requires the robot arm to reach the right button on the stovetop and press it. Partial success is given if the robot is within range of the button but does not touch it.
- **Close Microwave.** This task requires the robot to close the microwave door. The angle at which the microwave door is open varies between evaluation rollouts. Partial success is given if the robot pushes the microwave, but does not completely close the microwave, which upon closing properly will produce a click sound.
- **Put Green Block in Pot and Slide Pot Right.** This task requires the robot to first pick up the green block, drop it off at the pot, and then slide the pot right with its gripper. Partial success is provided for successfully completing a subtask of picking up the green block.

Robot Policy. For our policy, we are inspired by the architectural components introduced in Wang et al. [295] and Zhao et al. [338]. A diagram of our policy architecture is shown in Figure C.2. For both external and over-the-shoulder RGB images, we use a pretrained ResNet to first extract 7×7 feature maps and flatten these features across the spatial dimension to create a sequence of d_v dimension tokens where d_v is the output dimension of ResNet. In particular,

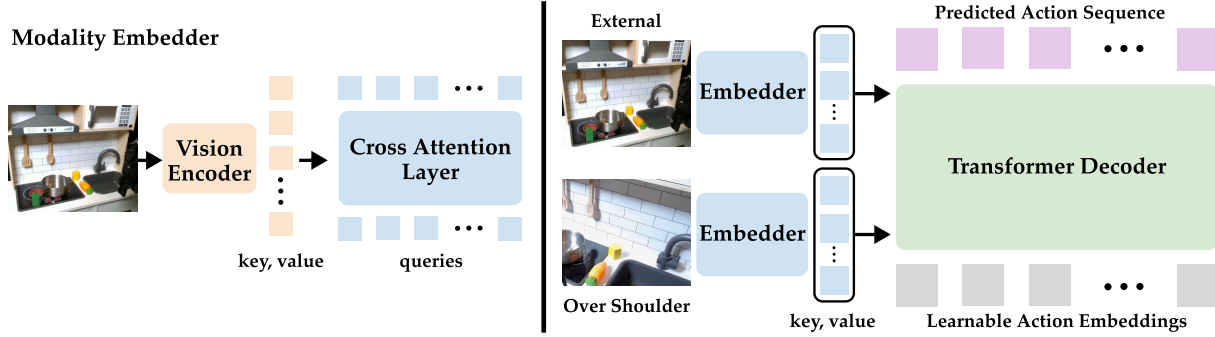


Figure C.2: Real Robot BC Policy. (Left) Learnable image embeddings following [295]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [338]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.

we use ResNet18 where $d_v = 512$. We feed as input to a causal transformer decoder a sequence learnable action tokens with dimension d . We use the flattened image feature map as the keys and values and apply a cross-attention between the image features and learnable tokens. We concatenate all modality tokens and add additional modality-specific embeddings and sinusoidal positional embeddings.

The policy base is a transformer decoder similar to the one used in ACT [338]. The input sequence to the transformer is a fixed position embedding, with dimensions $k \times 512$ where k is the chunk size and the keys and values are the combined image tokens from the stem. Given the current observation, we predict a chunk of 5 actions, which corresponds to 1 second of execution. During inference time, we also apply temporal ensembling similar to [338] with $m = 0.5$, which controls how much we consider older actions.

We train the policy for 20k update steps with batch size of 256 and a learning rate of $3e^{-4}$ (around 2 hours of wall time). For behavior cloning policies, the action dimension is 7 comprising of the robot joint pose and the gripper state. For CLAM, we predict *latent actions* which we then decode to robot actions using the learned action decoder. Figure C.3 shows example real-world task rollouts for the learned policies.

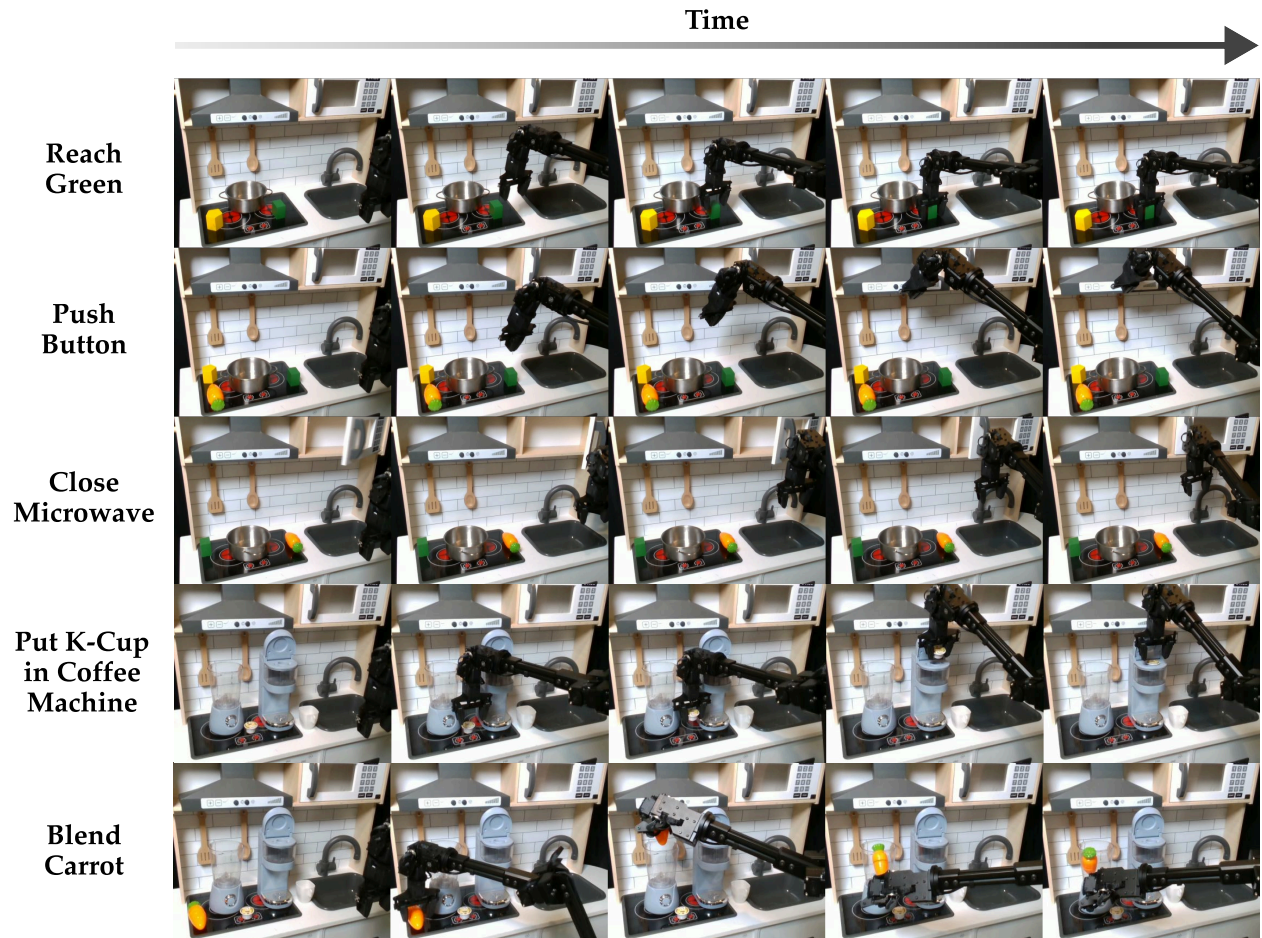


Figure C.3: Real Robot Evaluation Rollouts. We evaluate CLAM on four manipulation tasks. Each row is an example evaluation rollout and we subsample representative frames interpolated between the first and last timestep.

C.7 Architecture Details

We summarize the model architectures below using PyTorch-like notation. We also summarize the parameter counts of individual components for each method. Note, we share the same architecture between baseline and CLAM when possible to compare only the algorithms.

C.7.1 Behavior Cloning Policy

We use an MLP for our latent action policy in simulation and [53] in our real robot experiments. We use the same policy architecture across all the baseline methods to keep the results compa-

Method	Parameter Count (State/Image)
BC	1.6M
Latent Action Policy	1.4M
LAPO	Action Decoder (1M), LAM (9M/13M)
LAPA	IDM/FDM (9M/13M), Action Head (1M)
VPT	IDM (5M), Action Head (1M)
DynaMo	IDM/FDM (9M/13M)
State/Image Transformer CLAM	Action Decoder (1M), LAM (9M/13M)

table. For state-based experiments, we implement the policy as a 3-layer MLP with LeakyReLU activations. For image-based experiments, we additionally embed the image observation with a IMPALA-CNN [81] which is inputted into the MLP head to predict the final action. Finally, we apply a Tanh output activation to the model output to clip the action into a valid range, $[-1, 1]$.

BC MLP policy (state MetaWorld)

```
Total parameters: 1602567
Architecture: BC MLP Policy(
  (input_embedding): Linear(in_features=39, out_features=512, bias=True)
  (policy): Sequential(
    (0): Linear(in_features=512, out_features=1024, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=1024, out_features=1024, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=1024, out_features=4, bias=True)
  )
  (action_activation): Tanh()
)
```

C.7.2 Space Time CLAM

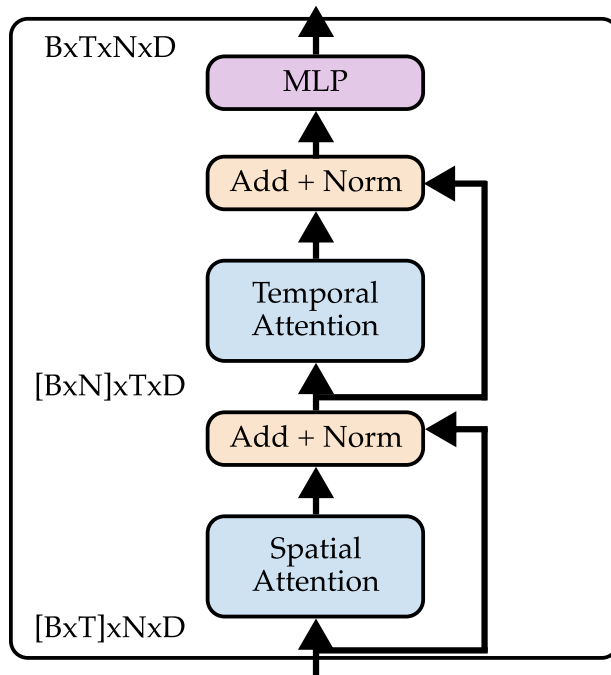


Figure C.4: Space-Time Attention Encoder Block Model architecture for the Space-Time Attention Encoder Block.

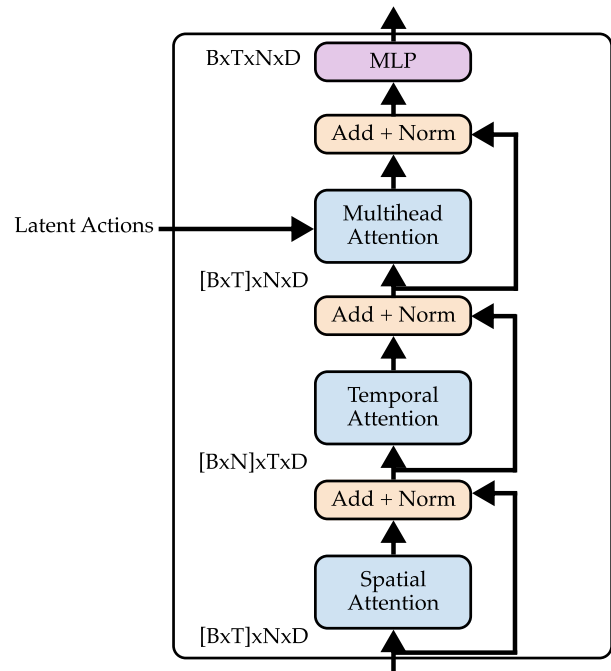


Figure C.5: Space-Time Attention Decoder Block Model architecture for the Space-Time Attention Decoder Block. Additional multihead attention to condition the learned representation on the latent action outputs from the IDM.

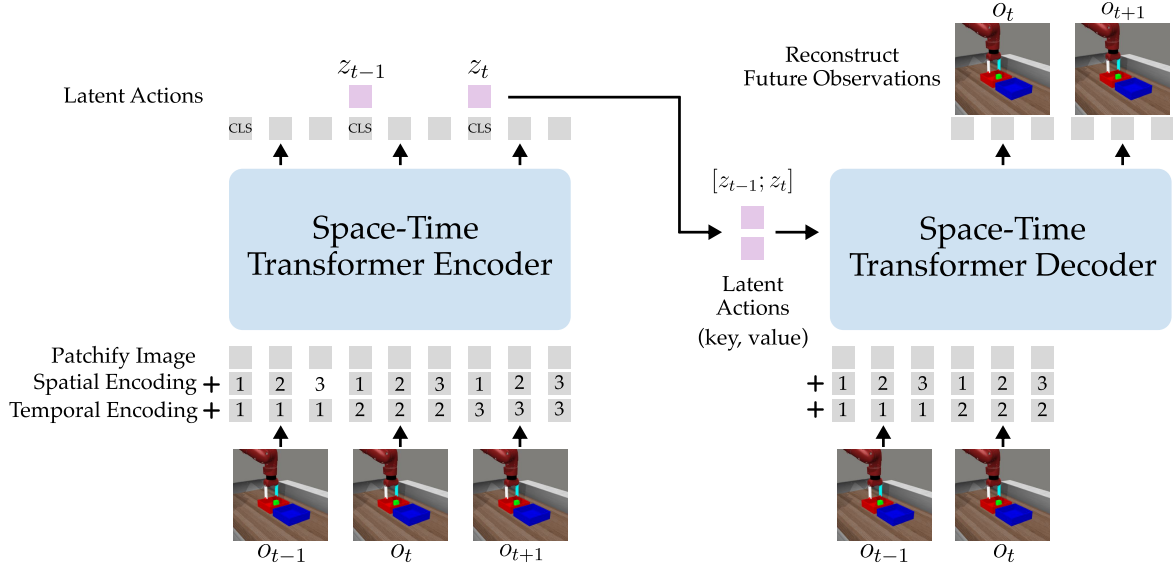


Figure C.6: Space-Time CLAM. Model architecture for the Space-Time Latent Action Model.

C.7.3 Latent Action Model.

For state-based experiments, we experiment with both an MLP-based and Transformer-based LAM. In the MLP-based variant, both the IDM and FDM are 3-layer MLPs with 1024 hidden dimensions. We use a context length of 1, thus the input to the IDM is $3 * T, (o_{t-1}, o_t, o_{t+1})$ and the input to the FDM is $2 * T + LA, (o_{t-1}, o_t, z_t)$ where T is the maximum number of episode steps, A is the action space dimension, S is the state space dimension, and LA is the latent action dimension.

For the transformer encoder, we first project the state input with a linear layer followed by several multihead self-attention layers and a final linear layer to predict the latent actions.

$$h_{t-1:t+1} = \text{TransformerEncoder}([o_{t-1}, o_t, o_{t+1}])$$

$$z_{t-1:t} = \text{LatentActionHead}(h_{t:t+1})$$

$$o_{t:t+1} = \text{TransformerDecoder}(h_{t-1:t}, z_{t-1:t})$$

The transformer decoder is architecturally similar to the encoder, except we apply a causal mask on the self-attention to prevent attending to the future states for the reconstruction. We additionally apply cross-attention to the latent actions predicted by the encoder without any

masking such that each embedding can attend to each latent action, even ones in a future timestep.

SpaceTime CLAM. For image observations, we model after the Space-Time(ST) Transformer [26]. We first patchify a $64 \times 64 \times 3$ image with a patch size of 16 for a total of 16 patches. Each patch is embedded through a linear layer into the hidden dimension. The encoder consists of N_E layers of Space-Time (ST) Attention blocks. Each ST block consists of spatial attention followed by temporal attention and a feedforward layer with skip connection, LayerNorm, and dropout applied between each attention.

The decoder ST block also applies a cross-attention with the latent actions generated by the encoder. A detailed illustration of an ST encoder/decoder block is shown in Figure C.4 and Figure C.5. We use a learned positional encoding which applies an `nn.Embedding` layer on the timestep indices and another to encode the index of each patches. We add an additional token in the sequence of patch embeddings as a CLS token for the whole image. From the CLS token for each timestep, we apply a linear layer to predict the latent actions. Figure C.6 summarizes the full architecture diagram for our Space-Time CLAM.

Below we provide pseudocode detailing how we apply ST attention to the patchified images.

Space-time attention (patch sequence)

```
patches = einops.rearrange(patches, "BTND->(BT)ND")
embedding = spatial_attn(patches)
embedding = einops.rearrange(embedding, "(BT)ND->(BN)TD")
embedding = temporal_attn(embedding)
```

Transformer CLAM.

Transformer CLAM (module summary)

```
Total Parameters: 8,874,551
Architecture: TransformerCLAM(
  (idm): TransformerIDM(
```

```

(input_embed): Linear(in_features=S, out_features=256, bias=True)
(activation): LeakyReLU(negative_slope=0.2)
(encoder): Encoder(
  (layers): ModuleList(
    (0-2): 3 x EncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True
        )
      )
      (linear1): Linear(in_features=256, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=256, bias=True)
      (norm1): LayerNorm((256,)), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((256,)), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
    )
  )
  (norm): Identity()
)
(latent_action): Linear(in_features=256, out_features=LA, bias=True)
(pos_embed): Embedding(T, 256)
)
(fdm): TransformerFDM(
  (activation): LeakyReLU(negative_slope=0.2)
  (input_embed): Linear(in_features=S, out_features=256, bias=True)
  (la_embed): Linear(in_features=LA, out_features=256, bias=True)
  (decoder): Decoder(
    (layers): ModuleList(
      (0-2): 3 x DecoderLayer(
        (self_attn): MultiheadAttention(

```

```

        (out_proj): NonDynamicallyQuantizableLinear(
            in_features=256, out_features=256, bias=True
        )
    )
    (multihead_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
            in_features=256, out_features=256, bias=True
        )
    )
    (linear1): Linear(in_features=256, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
    )
    )
    (norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    )
    (decoder_pos_embed): Embedding(T, 256)
    (encoder_pos_embed): Embedding(T, 256)
    (to_observation): Linear(in_features=256, out_features=S, bias=True)
    )

```

Space-Time CLAM

Space-Time CLAM (module summary)

Total Parameters: 12163344

```
(idm): SpaceTimeIDM(  
  (input_embed): Linear(in_features=768, out_features=256, bias=True)  
  (encoder): STTransformer(  
    (layers): ModuleList(  
      (0-2): 3 x STBlock(  
        (spatial_attn): MultiheadAttention(  
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features  
=256, bias=True)  
        )  
        (temporal_attn): MultiheadAttention(  
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features  
=256, bias=True)  
        )  
        (cross_attn): MultiheadAttention(  
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features  
=256, bias=True)  
        )  
        (linear1): Linear(in_features=256, out_features=2048, bias=True)  
        (dropout): Dropout(p=0.1, inplace=False)  
        (linear2): Linear(in_features=2048, out_features=256, bias=True)  
        (dropout1): Dropout(p=0.1, inplace=False)  
        (dropout2): Dropout(p=0.1, inplace=False)  
        (dropout3): Dropout(p=0.1, inplace=False)  
        (dropout4): Dropout(p=0.1, inplace=False)  
      )  
    )  
    (norm): Identity()  
  )  
  (activation): LeakyReLU(negative_slope=0.2)  
  (spatial_pos_embed): Embedding(200, 256)
```

```

(temporal_pos_embed): Embedding(200, 256)
(la_head): Linear(in_features=256, out_features=LA, bias=True)
)
(fdm): SpaceTimeFDM(
  (decoder): STTransformer(
    (layers): ModuleList(
      (0-2): 3 x STBlock(
        (spatial_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
        )
        (temporal_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
        )
        (cross_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
        )
        (linear1): Linear(in_features=256, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=256, bias=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
        (dropout4): Dropout(p=0.1, inplace=False)
      )
    )
    (norm): Identity()
  )
(patch_embed): Linear(in_features=768, out_features=512, bias=True)
(la_embed): Linear(in_features=16, out_features=512, bias=True)

```

```

(spatial_pos_embed): Embedding(200, 512)
(temporal_pos_embed): Embedding(200, 512)
(cond_pos_embed): Embedding(200, 512)
(to_recon): Linear(in_features=512, out_features=768, bias=True)
)

```

C.8 Data Generation

TD-MPC [112] is a model-based reinforcement learning (RL) algorithm that performs local trajectory optimization in the latent space of a learned world model. TD-MPC2 [113] provides a series of improvements over TD-MPC2 including architectural modifications and other design choices. All components of TD-MPC2 are implemented as MLPs with linear layers followed by LayerNorm and Mish activations. TD-MPC2 uses an ensemble of 5 Q-functions to to reduce bias in the TD-targets. It learns a closed-loop control policy by planning using a learned world model.

For our dataset collection, we train a TD-MPC2 agent for each MetaWorld environment from scratch with the default hyperparameter settings which can be found in Appendix H of the original paper. We report the episode returns as a function of environment steps in Figure C.7. We train each agent for 1M environment steps as we find that to be sufficient for the agent to learn to solve the task consistently. We store the low-dimensional state and RGB image observations from the replay buffer, which consist trajectories of varying expertise from a random uninitialized policy to a fully trained performant policy. Our final offline dataset consists of 1000 trajectories of 100 timesteps across four different MetaWorld tasks.

C.9 Baseline Details

We use the following official implementation repos for reference:

- Vector Quantization: <https://github.com/lucidrains/vector-quantize-pytorch>
- LAPO: <https://github.com/schmidtdominik/LAPO>

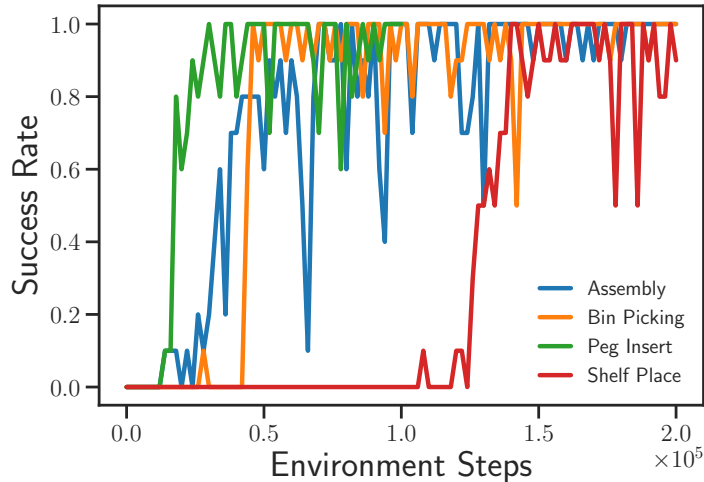


Figure C.7: Task Success Rate for Single-Task TD-MPC2 Agents. We train single-task agents using TD-MPC2 [113] to generate an offline dataset with different levels of behavior. All four agents exhibit expert performance on their respective tasks by the end of training.

- LAPA: <https://github.com/LatentActionPretraining/LAPA>
- GENIE: <https://github.com/myscience/open-genie>

We base our transformer encoder/decoder implementation on the `robot-transformers` repository at <https://github.com/KhaledSharif/robot-transformers/tree/main>.

We keep the architectures consistent across each method for fair comparison.

LAPO experiments on the Procgen [60] benchmark, which is a suite of 2D platformer games all of which have discrete action spaces. LAPO uses an EMA-based update for the vector quantization embedding and also a single step of additional context $H = 1$. We use the same hyperparameters for implementing VQ. Furthermore, they use a IMPALA-CNN for the encoder and decoder of the IDM and FDM respectively when working with image-based inputs. Refer to Appendix A.4 in their paper for a complete list of hyperparameters.

LAPA follows a similar implementation as LAPO. We reuse the same VQ implementation to discretize the learned latent actions. Unlike LAPA, we do not train our latent policy with language-conditioned data, which we will reserve for future work. After latent policy training, LAPA discards the latent action prediction head and fine-tunes the backbone model with a new action prediction head/ We implement the action head as an additional linear layer on top of the

pretrained backbone. LAPA requires finetuning on expert trajectories.

DynaMo. We implement DynaMo’s dynamics loss on the future latent embedding prediction and covariance regularization objectives following their open-sourced codebase. We tried both using a frozen ResNet-18 and training the image encoder from scratch. Following the paper, we apply causal masking in both the transformer encoder and decoder. Note for CLAM, we do not apply causal masking in the encoder and we allow the model to attend to the full sequence of input.

MLP experiments use Tesla V100s and Transformer experiments use NVIDIA A100.

Compute used for training on MetaWorld:

- BC (states): 2 hours, (images): 4 hours
- LAPA pretraining: (states): 2 hours, (images): 4.5 hours
- LAPO pretraining: (states): 3 hours pretraining, (images): 4.5 hours
- VPT IDM pretraining: (states): 1 hour, (images): 4.5 hours
- DynaMo: (states) 2 hours, (images) 4.5 hours
- Transformer CLAM (states): 2 hours
- SpaceTime Transformer CLAM (images): 4.5 hours
- Latent Policy Training (states): 2 hours, (images): 4 hours

All experiments use NVIDIA RTX A6000.

Compute used for training on CALVIN:

- BC (states): 1 hour
- VPT IDM pretraining (states): 4 hours
- Transformer CLAM (states): 4.5 hours
- Latent Policy Training (states): 2 hours

Appendix D

Inference-Time Plan Guidance

D.1 Training and inference algorithms

D.1.1 PLANTAIN Algorithm

Algorithm 12 PLANTAIN: Post-training to Elicit Interleaved Reasoning

Require: Base LM π_{θ_0} ; reference policy π_{ref} ; value net V_ϕ ; prompts $\mathcal{X}$; SI template $\text{SI}_{\text{plan-first}}$; synthetic generator Π^* (larger LM); weights $\alpha_{\text{fmt}}, \alpha_{\text{corr}}, \alpha_{\text{help}}, \alpha_{\text{unit}}$; PPO coeff β

Ensure: Interleaved model π_θ (plan-first, then alternating thoughts/answers)

Synthetic Interleaved Trace Dataset

- 1: $\mathcal{D}_{\text{interleave}} \leftarrow \emptyset$
- 2: **for all** $x \in \mathcal{X}$ **do**
- 3: $x' \leftarrow \text{SI}_{\text{plan-first}}(x)$
- 4: $\tau = (t_1, a_1, t_2, a_2, \dots, t_n, a_n) \sim \Pi^*(\cdot \mid x')$ $\triangleright a_1$ is the explicit *plan*
- 5: $\mathcal{D}_{\text{interleave}} \leftarrow \mathcal{D}_{\text{interleave}} \cup \{(x', \tau)\}$

Supervised Fine-tuning

- 6: $\theta \leftarrow \arg \min_{\theta} \sum_{(x', \tau) \in \mathcal{D}_{\text{interleave}}} (-\log \pi_\theta(\tau \mid x'))$ $\triangleright$ Shift style from monolithic to interleaved

RL Post-Training (PPO)

- 7: **repeat**
 - 8: Sample $(x', \cdot) \sim \mathcal{D}_{\text{interleave}}$; rollout $y \sim \pi_\theta(\cdot \mid x')$
 - 9: $r_{\text{fmt}} \leftarrow \text{FORMATOK}(y)$ $\triangleright$ plan-first; valid interleaving; required sections
 - 10: $r_{\text{corr}} \leftarrow \text{UNITTESTPASSRATE}(x', y)$
 - 11: $r_{\text{help}} \leftarrow \text{LLMJUDGEHELPFULNESS}(x', y)$
 - 12: $r_{\text{unit}} \leftarrow \mathbf{1}\{\text{unit_tests present in } y\}$
 - 13: $g \leftarrow \mathbf{1}\{r_{\text{fmt}} = 1\}$ $\triangleright$ gate downstream rewards on correct interleaved format
 - 14: $r(x', y) \leftarrow \alpha_{\text{fmt}} r_{\text{fmt}} + g \cdot (\alpha_{\text{corr}} r_{\text{corr}} + \alpha_{\text{help}} r_{\text{help}} + \alpha_{\text{unit}} r_{\text{unit}})$
 - 15: Compute advantages $\hat{A}$ with GAE using V_ϕ ; update θ, ϕ via PPO:
 - 16:
$$\max_{\theta} \mathbb{E} \left[\text{clip} \left(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}, 1 \pm \epsilon \right) \hat{A} \right] - \beta \text{D}_{\text{KL}}(\pi_\theta(\cdot \mid x') \parallel \pi_{\text{ref}}(\cdot \mid x'))$$
 - 17: **until** convergence
 - 18: **return** π_θ
-

Algorithm 13 Inference Strategy 1: Best-of- N Plan

Require: Trained interleaved model π_θ ; prompt p ; temperature $\tau > 1$; number of plans N ; LLM Autorater

- 1: Form plan-first instruction $p' \leftarrow \text{SI}_{\text{plan-first}}(p)$
 - 2: Generate candidate plans $\mathcal{A} \leftarrow \{a_i \sim \pi_\theta(a \mid p'; \tau)\}_{i=1}^N$
 - 3: Select $i^* \leftarrow R_{\text{select}}(p, \mathcal{A})$; $a^* \leftarrow a_{i^*}$
 - 4: Roll out remainder conditioned on a^* : $y \sim \pi_\theta(\cdot \mid p', a^*)$
 - 5: **return** y
-

Algorithm 14 Inference Strategy 2: Rewind & Repeat - Iterative Plan Rejection Sampling

Require: Trained interleaved model π_θ ; prompt p ; maximum retries T ; LLM Autorater R_{bin}

- 1: $p' \leftarrow \text{SI}_{\text{plan-first}}(p)$; $\mathcal{H} \leftarrow \emptyset$
 - 2: **for** $t = 1$ to T **do**
 - 3: $c_t \leftarrow \text{AUGMENT}(p', \mathcal{H})$
 - 4: Propose plan $a_t \sim \pi_\theta(a \mid c_t)$
 - 5: $d_t \leftarrow R_{\text{bin}}(p, a_t)$
 - 6: **if** $d_t = \text{accept}$ **then**
 - 7: **return** $y \sim \pi_\theta(\cdot \mid c_t, a_t)$
 - 8: **else**
 - 9: $\mathcal{H} \leftarrow \mathcal{H} \cup \{a_t\}$
 - 10: Sample a random plan $a_r \sim \text{Uniform}(\mathcal{H})$
 - 11: **return** $y \sim \pi_\theta(\cdot \mid p', a_r)$
-

D.2 Prompts, datasets, and implementation

D.2.1 Interleaved Reasoning System Instruction

Interleaved reasoning (system instruction)

You are a helpful assistant. You reason through problems step-by-step before providing an answer. You conduct your reasoning within `<think></think>` and share partial answers that are useful for the user within `<answer></answer>`. You continue this pattern of `<think>...</think>` `<answer>...</answer>` `<think>...</think>` `<answer>...</answer>` until you reach the final answer.

User: {insert prompt here}

Assistant:

D.2.2 LLM Plan Autorater

LLM plan autorater

You are an expert plan evaluator. Given a prompt and a single plan, your task is to determine if the plan is good enough to proceed with.

Evaluation Criteria:

1. Completeness: Does the plan address all aspects of the prompt?
2. Feasibility: Is the plan realistic and implementable?
3. Clarity: Is the plan clear and well-structured?
4. Alignment: Does the plan align with the user's intent?

Carefully evaluate the plan based on the criteria above. Consider whether this plan provides a solid foundation for addressing the user's request.

A plan is APPROVED (Decision: TRUE) if it:

1. Directly addresses the core question or request
2. Provides clear, actionable steps
3. Covers the essential aspects without being overly complex
4. Aligns with the user's apparent intent

A plan is REJECTED (Decision: FALSE) if it:

1. Misses key aspects of the request
2. Is too vague or abstract to act on
3. Contains unrealistic or impractical elements
4. Is incomplete or poorly structured

Respond with exactly one line in this format: Decision: TRUE or Decision: FALSE

Please proceed with the evaluation. Decision:

Best-of-N plan selection

You are an expert plan evaluator. Given a prompt and several candidate plans, select the single plan that provides the best foundation for answering the prompt correctly.

Choose the plan that is most complete, feasible, clear, and aligned with the user’s intent. Prefer plans that address all important constraints in the prompt and are likely to lead to a correct final response.

Respond with exactly one line containing the selected plan index:

Selected: 1

Prompt: {insert prompt here}

Candidate plans: {insert indexed candidate plans here}

Selected:

D.2.3 Dataset Details

Table D.1: SFT dataset details.

Dataset	Size
BigCodeBench [347]	50
MBPP [11] (Concat-2)	50
MATH500 [191] (Concat-2)	50
Multiple Solutions	26

D.2.4 Training and Evaluation Datasets

SFT and RL Training Datasets. We use a combination of coding and mathematical reasoning datasets to train our interleaved reasoning models.

1. **BigCodeBench (BCB)** [347] is a benchmark of 1,140 Python programming tasks that require diverse function calls from common libraries such as `numpy` and `matplotlib`. Each task includes an average of 5.6 unit tests with 99% branch coverage. For training, we split

Table D.2: RL dataset details.

Dataset	Size
BigCodeBench [347]	500
MBPP [11] (Concat-2)	50
MATH500 [191] (Concat-2)	50
Multiple Solutions	26

the dataset into train and test sets, and randomly sample 50 prompts from the training split to generate synthetic interleaved responses. To better suit interleaved reasoning, we modify the original prompts to include not only code generation but also a brief solution outline and associated unit tests.

2. **Mostly Basic Python Programs (MBPP)** [11] consists of 974 crowd-sourced Python problems designed to be solvable by entry-level programmers. Each problem includes a short text description and three test cases. To promote multi-step reasoning, we sample multiple problems and concatenate them into a single composite prompt that requires the model to solve each sequentially.
3. **MATH500** [191] contains 500 diverse math problems spanning topics such as probability, algebra, trigonometry, and geometry. Similar to MBPP, we randomly combine multiple problems into a single prompt to encourage multi-stage reasoning and plan refinement.

Evaluation Datasets. We evaluate the trained models across domains that require coding, mathematical reasoning, symbolic translation, and long-context comprehension.

- **BigCodeBench (BCB)** [347]: challenging Python coding prompts requiring composition across multiple libraries.
- **Mostly Basic Python Programs (MBPP)** [11]: simple Python programming tasks solvable by entry-level programmers.
- **MATH500** [191]: diverse mathematical reasoning problems across algebra, probability, and geometry.
- **BirdSQL** [181]: cross-domain text-to-SQL benchmark with over 12k question–SQL pairs

across 95 databases.

- **QuaLITY** [235]: long-document multiple-choice QA benchmark designed to test reasoning over extended contexts.

D.2.5 Training Details

Table D.3: Training hyperparameters.

Parameter	Value
Actor learning rate	1×10^{-6}
Critic learning rate	1×10^{-6}
Train batch size per gpu	32
Validation batch size	256
PPO mini batch size	32
PPO micro batch size	16
Critic micro batch size	8
KL coefficient	0.001
KL loss type	low variance KL
Max prompt length	3096 tokens
Max response length	2500 tokens
Sampling temperature	0.7
Number of samples per prompt	8
Stable training threshold (ϵ)	0.05
Critic warmup steps	0
Evaluation frequency	50 steps
Tensor model parallel size	2

Experiments were conducted building on VERL [263], an efficient reinforcement learning framework for post-training language models. We performed all experiments on 8 NVIDIA H100 GPUs with 80GB memory in a Google Cloud VM. We also used a consistent set of hyperparameters to ensure fair comparison between methods. We evaluate and save every 50 steps during training, and continue training from the last saved checkpoint if the training is interrupted (e.g., OOM).

D.2.6 Synthetic Interleaved Response Generation

We generate synthetic interleaved reasoning traces by iteratively prompting a larger model (Qwen3-32B). For BigCodeBench, we modify the vanilla coding prompts, to also ask for a solution outline and unit tests in addition to the code solution. We prompt the model to generate thought and intermediate response traces in this order: `thought` $\rightarrow$ `code` `outline` $\rightarrow$ `thought` $\rightarrow$ `code` `solution` $\rightarrow$ `thought` $\rightarrow$ `unit tests`. To prevent the base model from spending all its output budget on thinking, we terminate the thought generation after $N = 256$ tokens, clean up the final sentence, and append a `</think>` token to mark the end of a subthought. Then we reprompt the model provided the previously generated thoughts as context to generate the next intermediate response. We provide an example of this interleaved trace at Example [D.3.1](#) below.

D.3 Additional results and examples

D.3.1 Training dataset prompt examples

BCB-Outline-Code-Unit

Question:

Save the provided Pandas DataFrame “df” in a pickle file with the given name, read it back for validation, and delete the intermediate file. The function should output with: loaded_df (pd.DataFrame): The loaded DataFrame from the specified file. You should write self-contained code starting with:

```
import pickle
import os

def task_func(df, file_name="save.pkl"):
    # function implementation
    pass
```

First, outline the solution in a markdown format. Then, write the code to implement the solution. Finally, generate unit tests to test the code. Format the unit tests as a python function with a docstring. Use this exact format:

```
import unittest
from task_func import task_func
class Test(unittest.TestCase):
    def test_case_1(self):
        # Test case 1 description
        result = task_func(...)
        self.assertEqual(result,
                          expected_value)
```

MATH500 Concatenate 2

Question:

Solve the following problems:

- 1) Find all p which satisfy both the inequalities $0 \geq 54p - 144$ and $0 > 12 - 20p$. Express your answer in interval notation, reducing any fractions in your answer.
- 2) A strictly increasing sequence of positive integers $a_1, a_2, a_3, \dots$ has the property that for every positive integer k , the subsequence $a_{2k-1}, a_{2k}, a_{2k+1}$ is geometric and the subsequence $a_{2k}, a_{2k+1}, a_{2k+2}$ is arithmetic. Suppose that $a_{13} = 2016$. Find a_1 .

Answer:

<think> Okay, so I need to find all values of p that satisfy both inequalities: $0 \geq 54p - 144$ and $0 > 12 - 20p$. Hmm, let me start by solving each inequality separately and then find their intersection ... </think>

<answer> $\left(\frac{3}{5}, \frac{8}{3}\right]$ </answer>

<think> Okay, so I need to solve this problem about a strictly increasing sequence of positive integers. The sequence has some properties related to geometric and arithmetic subsequences. ...

</think>

<answer> 504 </answer>

MBPP Concatenate 2

Question:

Solve the following coding problems:

- 1) Write a function to reverse words in a given string.
- 2) Write a function that matches a string that has an `a` followed by zero or more `b`'s by using regex. You should write self-contained code for each problem starting with: `def task_func(args):`

Answer:

<think> Okay, I need to write a function that reverses the words in a given string. Let me think about how to approach this. So, the input is a string like "Hello world", and the output should be "world Hello"... </think>

<answer>

```
def task_func(s):  
    return ' '.join(reversed(s.split()))
```

</answer>

<think> Okay, I need to write a function that checks if a string has an 'a' followed by zero or more 'b's using regex. Let me think about how to approach this. First, I remember that regular expressions use certain patterns. The problem says "a followed by zero or more b's". So the 'a' must be there, and then after it, there can be any number of 'b's, including none... </think>

<answer>

```
import re  
def task_func(text):  
    patterns = 'ab*?'  
    if re.search(patterns, text):  
        return ('Found a match!')  
    else:  
        return ('Not matched!')
```

</answer>

Multiple Code Solutions

Question:

Implement a function that performs basic string compression (e.g., 'aabccccaaa' becomes 'a2b1c5a3'). Provide an iterative solution and one using itertools.

You should write self-contained code for each problem starting with: `def task_func(args):`

Answer:

<think> Okay, I need to write an iterative Python function for string compression. Let's see. The problem says that consecutive repeated characters should be replaced by the character followed by the count. And if the compressed string isn't shorter, return the original. First, I should think about how to process the string. Since it's iterative, I can't use recursion... </think>

<answer>

```
def task_func(s: str) -> str:
    if not s:
        return s

    compressed = []
    current_char = s[0]
    count = 1

    for char in s[1:]:
        if char == current_char:
            count += 1
        else:
            compressed.append(
                current_char + str(count)
            )
            current_char = char
            count = 1

    # Append the last character
    # and its count
    compressed.append(current_char
        + str(count))

    compressed_str = ''.join(compressed)
    return compressed_str

    if len(compressed_str) < len(s)
    else s
```

</answer>

<think> Okay, I need to write a Python function called `task_func` that compresses a string using `itertools`. The goal is to replace consecutive repeated characters with the character followed by

the count. And if the compressed version isn't shorter, return the original. First, I remember that itertools has groupby, which groups consecutive elements ... </think>

<answer>

```
from itertools import groupby

def task_func(s: str) -> str:
    compressed = ''.join(char
        + str(len(list(group)))
        for char, group in groupby(s))
    return compressed
    if len(compressed) < len(s)
    else s
```

</answer>

D.3.2 Generated Plan and Answers

MATH500

Question:

Convert the point $(0, 3)$ in rectangular coordinates to polar coordinates. Enter your answer in the form (r, θ) , where $r > 0$ and $0 \leq \theta < 2\pi$.

Plan:

1. Identify the rectangular coordinates $(x, y) = (0, 3)$.
2. Use the formula $r = \sqrt{x^2 + y^2}$ to calculate the radial distance r .
3. Determine the angle θ using the relationship $\theta = \arctan\left(\frac{y}{x}\right)$, adjusting for the correct quadrant based on the signs of x and y .
4. For the point $(0, 3)$, recognize that it lies on the positive y -axis, so $\theta = \frac{\pi}{2}$.
5. Verify that θ is within the required range $0 \leq \theta < 2\pi$.
6. Write the final polar coordinates in the form (r, θ) .

Answer: $(3, \frac{\pi}{2})$

BirdSQL

Database: debit_card_specializing

Table: customers

CustomerID	identification of the customer (integer)
Segment	client segment (text)
Currency	Currency (text)

Table: gasstations

GasStationID	Gas Station ID (integer)
ChainID	Chain ID (integer)
Country	No description (text)
Segment	chain segment (text)

Table: products

ProductID	Product ID (integer)
Description	Description (text)

Table: transactions_1k

TransactionID	Transaction ID (integer)
Date	Date (date)
Time	Time (text)
CustomerID	Customer ID (integer)
CardID	Card ID (integer)
GasStationID	Gas Station ID (integer)
ProductID	Product ID (integer)
Amount	Amount (integer)
Price	Price (real)

Note: total price = Amount $\times$ Price

Table: yearmonth

CustomerID	Customer ID (integer)
Date	Date (text)
Consumption	consumption (real)

Question: In 2012, who had the least consumption in LAM?

Plan:

1. Filter the `yearmonth` table for the year 2012 by checking if the 'Date' starts with '2012'.
2. Join the `yearmonth` table with the `gasstations` table using the `GasStationID` to retrieve country information.
3. Filter the joined dataset to include only rows where Country is LAM.
4. Aggregate the `Consumption` values per customer using `GROUP BY CustomerID`.
5. Sort the results in ascending order of the total consumption.
6. Use `LIMIT 1` to retrieve the customer with the least consumption.

Answer:

```
SELECT y.CustomerID, SUM(y.Consumption) AS total_consumption
FROM yearmonth y
JOIN gasstations g ON y.GasStationID = g.GasStationID
WHERE LEFT(y.Date, 4) = '2012' AND g.Country = 'LAM'
GROUP BY y.CustomerID
ORDER BY total_consumption ASC
LIMIT 1;
```

D.3.3 MATH500 Think-Answer vs. Rewind-and-Repeat

MATH500 Problem 153 Rewind-and-Repeat

Question: The medians AD , BE , and CF of triangle ABC intersect at the centroid G . The line through G that is parallel to BC intersects AB and AC at M and N , respectively. If the area of triangle ABC is 144, then find the area of triangle ENG .

<think> ... </think>

<answer>

1. Use coordinate geometry to assign coordinates to triangle ABC , simplifying calculations.
2. Determine the coordinates of the centroid G using the midpoint formula.
3. Find the equation of the line through G that is parallel to BC .
4. Calculate the coordinates of points M and N , where this line intersects AB and AC , respectively.
5. Use the coordinates of points E , N , and G to compute the area of triangle ENG using the shoelace formula or determinant method.
6. Relate the area of triangle ENG to the area of triangle ABC , which is given as 144, to find the numerical value.

</answer>

<think> Okay, let's start by assigning coordinates to triangle ABC . I'll place point A at $(0, 0)$, B at $(2b, 0)$, and C at $(0, 2c)$. This makes AB and AC the axes, simplifying calculations. The centroid G is the average of the coordinates, so G should be at $(\frac{2b}{3}, \frac{2c}{3})$.

Next, I need the equation of the line through G parallel to BC . The slope of BC is $\frac{2c-0}{0-2b} = -\frac{c}{b}$. So the line through G with this slope is $y - \frac{2c}{3} = (-\frac{c}{b})(x - \frac{2b}{3})$. Let me rearrange this equation.

(Intermediate reasoning steps omitted for brevity.)

MATH500 Problem 153 Rewind-and-Repeat (continued)

So triangle ENG has points $E(0, c)$, $N(0, \frac{4c}{3})$, and $G(\frac{2b}{3}, \frac{2c}{3})$. Let's compute its area. Using the shoelace formula:

Area = $\frac{1}{2}|(x_E(y_N - y_G) + x_N(y_G - y_E) + x_G(y_E - y_N))|$. Plugging in:

$$x_E = 0, y_E = c; x_N = 0, y_N = \frac{4c}{3}; x_G = \frac{2b}{3}, y_G = \frac{2c}{3}.$$

Calculating each term:

$$\text{First term: } 0 \cdot (\frac{4c}{3} - \frac{2c}{3}) = 0 \cdot (\frac{2c}{3}) = 0.$$

$$\text{Second term: } 0 \cdot (\frac{2c}{3} - c) = 0 \cdot (-\frac{c}{3}) = 0.$$

$$\text{Third term: } (\frac{2b}{3}) \cdot (c - \frac{4c}{3}) = (\frac{2b}{3}) \cdot (-\frac{c}{3}) = -\frac{2bc}{9}.$$

$$\text{Taking absolute value and multiplying by } \frac{1}{2}: \frac{1}{2} \cdot |-\frac{2bc}{9}| = \frac{1}{2} \cdot \frac{2bc}{9} = \frac{bc}{9}.$$

So the area of ENG is $\frac{bc}{9}$.

Now, relate this to the area of ABC . The area of ABC is 144. Since ABC is a right triangle with legs $2b$ and $2c$, its area is $(\frac{1}{2}) \cdot 2b \cdot 2c = 2bc$. So $2bc = 144 \Rightarrow bc = 72$.

Therefore, the area of ENG is $\frac{72}{9} = 8$.

Wait, that seems straightforward. Let me check each step again. The coordinates assignment, centroid calculation, line equation, points M and N , and area computation. All steps seem correct. The key was setting up the coordinate system and using the shoelace formula. The answer should be 8. </think>

<answer> 8 </answer>

MATH500 Problem 153 Think-Answer

Question: The medians AD , BE , and CF of triangle ABC intersect at the centroid G . The line through G that is parallel to BC intersects AB and AC at M and N , respectively. If the area of triangle ABC is 144, then find the area of triangle ENG .

<think> Okay, so I need to find the area of triangle ENG in triangle ABC , where the medians AD , BE , and CF intersect at centroid G . There's a line through G parallel to BC that intersects AB at M and AC at N . The area of ABC is 144. Hmm, let me try to visualize this first.

First, let me recall some properties of centroids. The centroid divides each median into a ratio of 2:1, with the longer part being closer to the vertex. So, for example, $AG : GD = 2 : 1$, $BG : GE = 2 : 1$, and $CG : GF = 2 : 1$.

Now, there's a line through G parallel to BC . Since BC is the base of the triangle, this line is going to be somewhere inside the triangle. It intersects AB at M and AC at N . So, MN is parallel to BC and passes through G . I need to figure out where exactly M and N are located on AB and AC .

(Intermediate reasoning steps omitted for brevity.)

MATH500 Problem 153 Think-Answer (continued)

Since MN is parallel to BC , triangle AMN is similar to triangle ABC by the Basic Proportionality Theorem or Thales' theorem. Wait, but MN isn't necessarily a midline unless it's halfway. However,

since it passes through the centroid, which divides the medians in a 2:1 ratio, maybe I can find the ratio of similarity.

Let me think. Let me consider coordinates. Maybe coordinate geometry can help here. Let me assign coordinates to triangle ABC . Let me place point A at $(0,0)$, B at $(2b,0)$, and C at $(2c,2d)$. Wait, but maybe it's better to use coordinates that make calculations easier. Since the centroid divides the medians in 2:1, maybe choosing coordinates where the centroid is at $(0,0)$? Wait, but maybe that complicates things. Alternatively, let me use standard coordinates.

Alternatively, let me take coordinates such that BC is horizontal for simplicity. Wait, but since MN is parallel to BC , maybe it's better to have BC horizontal. Let me try.

First, equation of AB : from $A(a,b)$ to $B(0,0)$. The parametric equations can be written as $x = a - a \cdot t$, $y = b - b \cdot t$, where t ranges from 0 to 1. Wait, or maybe better to use a parameter s .

Alternatively, let me find the equation of AB . The slope of AB is $\frac{0-b}{0-a} = \frac{b}{a}$. So equation is $y = (\frac{b}{a})x$. Similarly, equation of AC : from $A(a,b)$ to $C(c,0)$. The slope is $\frac{0-b}{c-a} = -\frac{b}{c-a}$. Equation is $y - b = [-\frac{b}{c-a}](x - a)$.

Now, the line $y = b/3$ intersects AB and AC . Let me find coordinates of M and N .

For point M on AB : $y = (\frac{b}{a})x = \frac{b}{3}$. Therefore, $x = (\frac{b}{3}) \cdot (\frac{a}{b}) = \frac{a}{3}$. Therefore, coordinates of M are $(\frac{a}{3}, \frac{b}{3})$.

Similarly, for point N on AC : Let me solve $y = \frac{b}{3}$ with equation of AC .

Equation of AC : $y = [-\frac{b}{c-a}](x - a) + b$.

Set $y = b/3$:

$$\frac{b}{3} = [-\frac{b}{c-a}](x - a) + b$$

...

Wait no. Wait no. Wait no. Wait no... </think>

Appendix E

DynaMITE-RL

E.1 ELBO derivation for DLCMDP

We will define a full trajectory $\tau = \{s_0, a_0, r_1, s_1, a_1, \dots, r_{H-1}, s_H\}$ where H is the horizon. $\tau_{:t}$ is the history of interactions up to a global timestep t , i.e. $\tau_{:t} = \{s_0, a_0, r_1, s_1, a_1, \dots, r_{t-1}, s_t\}$.

Let $\mathcal{Z} = \{m^0, \dots, m^{K-1}\}$ be the collection of latent contexts in a trajectory where K is a random variable representing the number of switches the latent variable will have until time H , i.e., $K = \sum_{t=0}^{H-1} d_t$. Additionally, we denote d_t as the session termination prediction at timestep t but $d_{H-1} \equiv 1$.

We divide a full trajectory into sessions and define a discrete random variable $t_i \in \{0, \dots, H-1\}$ be a random variable denoting the last timestep of session $i \in \{0, \dots, K-1\}$, i.e., $t_i = \min\{t' \in \mathbb{Z}_{\geq 0} : \sum_{t=0}^{t'} d_t = i + 1\}$, with $t_{-1} \equiv -1$. We also denote the next session index $i' = i + 1$.

An arbitrary session i' can then be represented as, $\{s_{t_i+1}, a_{t_i+1}, r_{t_i+1}, s_{t_i+2}, \dots, s_{t_{i'}-1}, a_{t_{i'}-1}, r_{t_{i'}}\}$.

At any time-step t , we want to maximize the log-likelihood of the full dataset of trajectories, $\mathcal{D}$, collected following policy π , e.g. $\mathbb{E}_\pi[\log p_\theta(\tau)]$. However, with the presence of latent variables, whose samples cannot be observed in the training data, estimating the empirical log-likelihood is generally intractable. Instead, we optimize for the evidence lower bound (ELBO) of this function with a learned approximate posterior, q_ϕ .

We then define the posterior inference model, $q_\phi(\mathcal{Z}, d_{:H} \mid \tau_{:t})$, which outputs the posterior

distribution for the latent context and session termination predictions conditioned on the trajectory history up until timestep t .

Below we provide the derivation for the variational lower bound of the log-likelihood function $\log p_\theta(\tau)$ for a single trajectory:

$$\begin{aligned}
\log p_\theta(\tau) &= \log \int_{\mathcal{Z}, \Omega} p_\theta(\tau, \mathcal{Z}, \Omega) \\
&= \log \int_{\mathcal{Z}, \Omega} p_\theta(\tau, \mathcal{Z}, \Omega) \frac{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})}{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \\
&= \log \mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left[\frac{p_\theta(\tau, \mathcal{Z}, \Omega)}{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \right] \\
&= \log \mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left[\frac{p_\theta(\tau \mid \mathcal{Z}, \Omega) p_\theta(\mathcal{Z}, \Omega)}{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \right] \\
&\geq \mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left[\log p_\theta(\tau \mid \mathcal{Z}, \Omega) + \log p_\theta(\mathcal{Z}, \Omega) - \log(q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})) \right] \\
&= \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left[\log p_\theta(\tau \mid \mathcal{Z}, \Omega) \right]}_{\text{reconstruction}} - \underbrace{D_{KL}(q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t}) \parallel p_\theta(\mathcal{Z}, \Omega))}_{\text{regularization}} \\
&= \text{ELBO}_t(\theta, \phi)
\end{aligned}$$

We extend this to derive the lower bound for all trajectories in dataset $\mathcal{D}$.

$$\mathbb{E}_{\tau \sim \mathcal{D}} \left[\log p_\theta(\tau) \right] = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left[\log p_\theta(\tau \mid \mathcal{Z}, \Omega) \right] - D_{KL}(q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t}) \parallel p_\theta(\mathcal{Z}, \Omega)) \right]$$

Prior:

$$p_\theta(\mathcal{Z}, \Omega) = p_\theta(m^0 \mid d_{:t_0}) p_\theta(d_{:t_0}) \prod_{i=0}^{K-2} p_\theta(m^{i'} \mid m^i, d_{t_i+1:t_{i'}}) p_\theta(d_{t_i+1:t_{i'}})$$

Variational Posterior:

$$q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t}) = q_\phi(m^0 \mid \tau_{:t_0}, d_{:t_0}) q_\phi(d_{:t_0}) \prod_{i=-1}^{K-2} q_\phi(m^{i'} \mid \tau_{t_i+1:t_{i'}}, m^i, d_{t_i+1:t_{i'}}) q_\phi(d_{t_i+1:t_{i'}})$$

Reconstruction Term:

$$\begin{aligned} \log p_\theta(\tau \mid \mathcal{Z}, \Omega) &= \log p_\theta(s_0, r_1, \dots, r_{H-1}, s_H \mid \mathcal{Z}, \Omega, a_{:H-1}) \\ &= \log \prod_{i=-1}^{K-2} p_\theta(\{(s_t, r_t)\}_{t=t_i+1}^{t_{i'}} \mid \mathcal{Z}, \Omega, a_{:H-1}) \\ &= \log \prod_{i=-1}^{K-2} \left[p_\theta(s_{t_i+1}) \prod_{t=t_i+1}^{t_{i'}} p_\theta(s_{t+1} \mid s_t, a_t, \mathcal{Z}, d_t) p_\theta(r_{t+1} \mid s_t, a_t, \mathcal{Z}, d_t) \right] \\ &= \sum_{i=-1}^{K-2} \left[\log p_\theta(s_{t_i+1}) + \sum_{t=t_i+1}^{t_{i'}} \log p_\theta(s_{t+1}, r_{t+1} \mid s_t, a_t, \mathcal{Z}, d_t) \right] \end{aligned}$$

Putting it all together:

$$\begin{aligned} \log p_\theta(\tau) &\geq \underbrace{\mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} [\log p_\theta(\tau \mid \mathcal{Z}, \Omega)]}_{\text{reconstruction}} - \underbrace{D_{KL}(q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})) \parallel p_\theta(\mathcal{Z}, \Omega)}_{\text{regularization}} \\ &= \mathbb{E}_{q_\phi(\mathcal{Z}, \Omega \mid \tau_{:t})} \left\{ \sum_{i=-1}^{K-2} \left[\log p_\theta(s_{t_i+1} \mid \mathcal{Z}, d_{t_i}) + \sum_{t=t_i+1}^{t_{i'}} \log p_\theta(s_{t+1}, r_{t+1} \mid s_t, a_t, \mathcal{Z}, d_t) \right] \right\} \\ &\quad - D_{KL}(q_\phi(m^0 \mid \tau_{:t_0}, d_{:t_0}) \parallel p_\theta(m^0 \mid d_{:H})) \\ &\quad - \sum_{i=0}^{K-2} D_{KL}(q_\phi(m^{i'} \mid \tau_{t_i+1:t_{i'}}, m^i, d_{t_i+1:t_{i'}}) \parallel p_\theta(m^{i'} \mid m^i, d_{t_i+1:t_{i'}})) \\ &\quad - \sum_{i=0}^{K-2} D_{KL}(q_\phi(d_{t_i+1:t_{i'}}) \parallel p_\theta(d_{t_i+1:t_{i'}})) \end{aligned}$$

E.2 Pseudocode for DynaMITE-RL

Here we provide the pseudocode for training DynaMITE-RL and for rolling out the policy during inference time.

Algorithm 15 DynaMITE-RL

```
1: Input: env,  $\alpha_\psi$ ,  $\alpha_\omega$ 
2: Randomly initialize policy  $\pi_\psi(a \mid s, m)$ , critic  $Q_\omega(s, a, m)$  decoder  $p_\theta(s', r' \mid s, a, m)$ , encoder  $q_\phi(m' \mid \cdot)$ ,
   and replay buffer  $\mathcal{D} = \emptyset$ 
3: for  $i = 1$  to  $N$  do
4:    $\mathcal{D}[i] \leftarrow \text{COLLECT\_TRAJECTORY}(\pi_\psi, q_\phi, \text{env})$ 
5:    $\triangleright$  Train VAE
6:   Sample batches of trajectories from  $\mathcal{D}$ 
7:   Compute ELBO with Eq. 7.4.2 and update  $\theta, \phi$ 
8:    $\triangleright$  Update actor and critic using PPO
9:    $\psi \leftarrow \psi - \alpha_\psi \nabla_\psi \mathcal{J}_\pi$ 
10:   $\omega \leftarrow \omega - \alpha_\omega \nabla_\omega \mathcal{J}_Q$ 
```

Algorithm 16 COLLECT_TRAJECTORY

```
1: Input:  $\pi_\theta, q_\phi, \text{env}$ 
2:  $(s_0, m_0) \sim v_0$  // sample initial state and belief
3:  $k = 0$  // session index
4: for  $t = 0$  to  $H - 1$  do
5:    $a_t \sim \pi_\psi(a_t \mid s_t, m_t)$  // get action
6:    $(s_{t+1}, r_{t+1}) = \text{env.step}(a_t)$  // env step
7:    $\triangleright$  Posterior update
8:   if  $k == 0$  then
9:      $m_{t+1}, d_{t+1} \sim q_\phi(\cdot \mid \tau_{:t+1}, d_t)$ 
10:  else
11:     $m_{t+1}, d_{t+1} \sim q_\phi(\cdot \mid \tau_{:t+1}, m_{t_{k-1}}, d_t)$ 
12:  if session-terminate then
13:     $k += 1$  // increment session index
14:     $(s_{t+1}, m_{t+1}) \sim v_0$  // reset the state
```

E.3 Additional experimental results

In this section, we report additional experimental results that did not fit in the main text, including per-episode performance over multiple rollouts and an ablation on the consistency weight hyperparameter β .

Following Zintgraf et al. [349], we measure test-time performance of meta-trained policies by evaluating per-episode return for 5 consecutive episodes, see Figure E.1. **DynaMITE-RL** and all of the baselines are designed to maximize reward *within a single rollout* hence they generally plateau after a single episode.

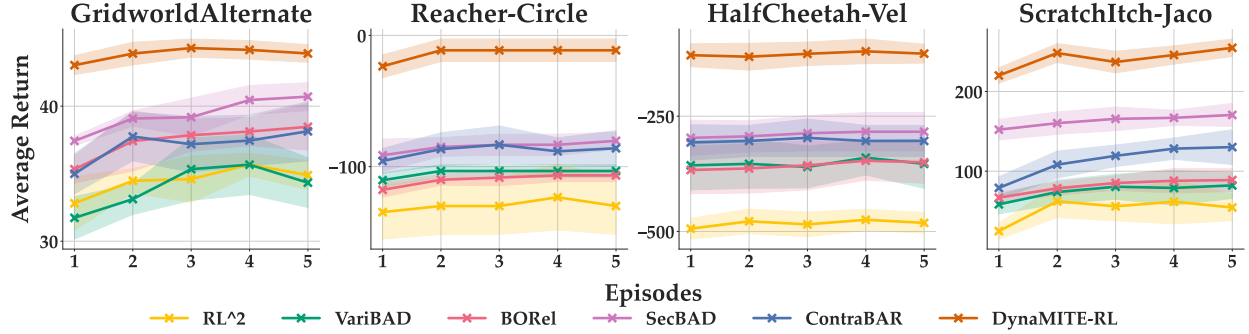


Figure E.1: Average test-time performance on MuJoCo tasks and ScratchItch task, trained separately with 5 seeds for MuJoCo tasks and 3 for itching task. The meta-trained policies are rolled out for 5 episodes to show how they adapt to the task. The returns averaged across the task with 95% confidence intervals shaded. We demonstrate that in our DLCMDP setting, the baseline methods struggle to adapt to the changing dynamics of the environment while our method learns the latent transitions and achieves good performance across all domains.

The β hyperparameter is a weight term for the consistency objective in DynaMITE-RL, which enforces an increase in information in subsequent timesteps. We run an ablation study over different values of β for the Half-Cheetah-Vel environment in our DLCMDP setting and find that in terms of final performance, our model is robust to the different value of β .

Table E.1: Ablation study over different values of β in the HalfCheetah-Vel environment.

β	Episode Return
0.01	-69.5 ± 2.6
0.1	-70.2 ± 2.5
1	-68.5 ± 2.3
5	-69.4 ± 3.0

E.4 Evaluation environments

In this section, we describe the details of the domains we used for our experiments. We provide visualizations of each simulation environment in Figure 7.4.

Gridworld Navigation with Alternating Goals

Following [349], we extend the 5×5 gridworld environment as shown in Figure 7.2. For each episode, two goal locations are selected randomly. However, only one of the goal locations provide a positive reward when the agent arrives at the location. The rewarding goal location changes between sessions according to some transition dynamics. In our experiments, we simulate latent dynamics using a simple transition matrix: $\begin{bmatrix} 0.2 & 0.8 \\ 0.8 & 0.2 \end{bmatrix}$. Between each session, the goal location has a 20% chance of remaining the same as the previous session and 80% chance of switching to the other location. The agent receives a reward of -0.1 on non-goal cells and +1 at the goal cell, e.g.

$$r_t = \begin{cases} 1 & \text{if } s_t = g \\ -0.1 & \text{otherwise} \end{cases}$$

where s_t is the current state and g is the current rewarding goal cell. Similar to [349], we set the maximum episode horizon to 60 and the Bernoulli probability for latent context switch to 0.25 such that in expectation each episode should have 4 sessions.

MuJoCo Continuous Control

For our study, we use the Brax [88] simulator, a physics engine for large scale rigid body simulation written in JAX. We use JAX [2], a machine learning framework which has just-in-time (jit) compilation that perform operations on GPU and TPU for faster training and can optimize the execution significantly. We evaluate the capacity of our method to perform continuous control tasks with high-dimensional observation spaces and action spaces.

Reacher is a two-joint robot arm task part of OpenAI’s MuJoCo tasks [39]. The goal is to move the robot’s end effector to a target 2D location. The goal locations change between each session following a circular path defined by: $[x, y] = [r \cos(\alpha \cdot i), r \sin(\alpha \cdot i)]$ where i is the session

index, $\alpha \sim \mathcal{U}(0, 2\pi)$ is the initial angle, and $r \sim \mathcal{U}(0.1, 0.2)$ is the circle's radius. The observation space is 11 dimensional consisting of information about the joint locations and angular velocity. We remove the target location from the observation space. The action space is 2 dimension representing the torques applied at the hinge joints. The reward at each timestep is based on the distance from the reacher's fingertip to the target: $r_t = -\|s_f - s_g\|_2 - 0.05 \cdot \|a_t\|_2$ where s_f is the (x, y) location of the fingertip and s_g for the target location.

Half-Cheetah builds off of the Half-Cheetah environment from OpenAI gym [39], a MuJoCo locomotion task. In these tasks, the challenge is to move legged robots by applying torques to their joints via actuators. The state space is 17-dimensional, position and velocity of each joint. The initial state for each joint is randomized. The action space is a 6-dimensional continuous space corresponding to the torque applied to each of the six joints.

Half-Cheetah Dir(ection): In this environment, the agent has to run either forward or backward and this varies between session following a transition function. At the first session, the task is decided with equal probability. The reward is dependent on the goal direction:

$$r_t = \begin{cases} v_t + 0.5 \cdot \|a_t\|_2 & \text{if task = forward} \\ -v_t + 0.5 \cdot \|a_t\|_2 & \text{otherwise} \end{cases}$$

where v_t is the current velocity of the agent.

Half-Cheetah Vel(ocity): In this environment, the agent has to run forward at a target velocity, which varies between sessions. The task reward is: $r_t = -\|v_s - v_g\|_2 - 0.05 \cdot \|a_t\|_2$, where v_s is the current velocity of the agent and v_g is the target velocity. The second term penalizes the agent for taking large actions. The target velocity varies between session according to: $v_g = 1.5 + 1.5\sin(0.2 \cdot i)$.

Half-Cheetah Wind + Vel: The agent is additionally subjected to wind forces which is applied to the agent along the x-axis. Every time the agent takes a step, it drifts by the wind vector. The force is changing between sessions according to: $f_w = 10 + 10 \sin(0.3 \cdot i)$.

Assistive Gym

Our assistive itch scratching task is adapted from Assistive Gym [80], similar to [279]. Assistive Gym is a simulation environment for commercially available robots to perform 6 basic activities of daily living (ADL) tasks - itch scratching, bed bathing, feeding, drinking, dressing, and arm manipulation. We extend the itch scratching task in Assistive Gym.

The itch scratching task contains a human and a wheelchair-mounted 7-DOF Jaco robot arm. The robot holds a small scratching tool which it uses to reach a randomly target scratching location along the human's right arm. The target location gradually changes along the right arm according to a predefined function, $x = 0.5 + \sin(0.2 \cdot i)$ where x is then projected onto a 3D point along the arm. Actions for each robot's 7-DOF arm are represented as changes in joint positions, $\mathbb{R}^7$. The observations include, the 3D position and orientation of the robot's end effector, the 7D joint positions of the robot's arm, forces applied at the robot's end effector, and 3D positions of task relevant joints along the human body. Again, the target itch location is unobserved to the agent.

The robot is rewarded for moving its end effector closer to the target and applying less than 10 N of force near the target. Assistive Gym considers a person's preferences when receiving care from a robot. For example, a person may prefer the robot to perform slow actions or apply less force on certain regions of the body. Assistive Gym computes a human preference reward, $r_H(s)$, based on how well the robot satisfies the human's preferences at state s . The human preference reward is combined with the robot's task success reward $r_R(s)$ to form a dense reward at each timestep, $r(s) = r_R(s) + r_H(s)$.

The full human preference reward is defined as:

$$r_H(s) = -\alpha \cdot \omega [C_v(s), C_f(s), C_{hf}(s), C_{fd}(s), C_{fdv}(s), C_d(s), C_p(s)]$$

where α is a vector of activations in $\{0, 1\}$ depicting which components of the preference are used and ω is a vector of weights for each preference category. $C.(s)$ is the cost for deviating from the human's preference.

$C_v(s)$ for high end effector velocities. $C_f(s)$ for applying force away from the target location. $C_{hf}(s)$ for applying high forces near the target (> 10 N). $C_{fd}(s)$ for spilling food or water. $C_{fdv}(s)$ for food / water entering mouth at high velocities. $C_d(s)$ for fabric garments applying force to the body. $C_p(s)$ for applying high pressure with large tools.

For our itch-scratching task, we set $\alpha = [1, 1, 1, 0, 0, 0, 0]$ and $\omega = [0.25, 0.01, 0.05, 0, 0, 0, 0]$.

E.5 Implementation details and hyperparameters

In this section, we provide the hyperparameter values used for training each of the baselines and DynaMITE-RL. We also provide more detailed explanation of the model architecture used for each method.

Online RL

We used Proximal Policy Optimization (PPO) training. The details of important hyperparameters used to produce the experimental results are presented in [Table E.2](#).

Table E.2: Training hyperparameters. Dashed entries means the same value is used across all environments.

	Gridworld	Reacher	HalfCheetah	ScratchItch
Max episode length	60	400	400	200
Bernoulli probability (p) for context switch	0.07	0.01	0.01	0.02
Number of parallel processes	16	2048	2048	32
Value loss coefficient	0.5	-	-	-
Entropy coefficient	0.01	0.05	0.05	0.1
Learning rate	3e-4	-	-	-
Discount factor (γ)	0.99	-	-	-
GAE lambda (λ_{GAE})	0.95	-	-	-
Max grad norm	0.5	-	-	-
PPO clipping epsilon	0.2	-	-	-
Latent embedding dimension	5	16	16	16
Policy learning rate	3e-4	-	-	-
VAE learning rate	3e-4	-	-	-
State/action/reward FC embed size	8	32	32	32
Consistency loss weight (β)	0.5	-	-	-
Variational loss weight (λ)	0.01	-	-	-

Table E.3: Hyperparameters for Transformer Encoder

Hyperparameter	Value
Embedding dimension	128
Num layers	2
Num attention head	8
Activation	GELU
Dropout	0.1

We also employ several PPO training tricks detailed in [123], specifically normalizing advantage computation, using Adam epsilon $1e - 8$, clipping the value loss, adding entropy bonus for better exploration, and using separate MLP networks for policy and value functions.

We use the same hyperparameters as above for RL² and VariBAD if applicable. For RL², the state and reward are embedded through fully connected (FC) layers, concatenated, and then

passed to a GRU. The output is fed through another FC layer and then the network outputs the actions.

ContraBAR: Code based on the author’s original implementation: <https://github.com/ec2604/ContraBAR> (MIT License). ContraBAR uses contrastive learning, specifically Contrastive Predictive Coding (CPC) [231], to learn an information state representation of the history. They use CPC to discriminate between positive future observations o_{t+k}^+ and K negative observations $\{o_{t+k}^-\}_{i=1}^K$ given the latent context c_t . The latent context is generated by encoding a sequence of observations through an autoregressive model. They apply an InfoNCE loss to train the latent representation.

DynaMITE-RL: The VAE architecture consists of a recurrent encoder, which at each timestep t takes as input the tuple (a_{t-1}, r_t, s_t) . The state, action, and reward are each passed through a different linear layers followed by ReLU activations to produce separate embedding vectors. The embedding outputs are concatenated, inputted through an MLP with 2 fully-connected layers of size 64, and then passed to a GRU to produce the hidden state. Fully-connected linear output layers generate the parameters of a Gaussian distribution: $(\mu(\tau_{:t}), \Sigma(\tau_{:t}))$ for the latent embedding m . Another fully-connected layer produces the logit for the session termination. The reward and state decoders are MLPs with 2 fully-connected layers of size 32 with ReLU activations. They are trained by minimizing a Mean Squared Error loss against the ground truth rewards and states. The policy and critic networks are MLPs with 2 fully-connected layers of size 128 with ReLU activations. For the domains where the reward function is changing between sessions, we only train the reward-decoder. For HalfCheetah Wind + Vel, we also train the transition decoder.

Offline RL

We use IQL [160] for offline RL training. IQL approximates the optimal value function through temporal difference learning by using expectile regression. IQL has a separate policy extraction step using advantage weighted regression (AWR) [238]. There are two main hyperparameters in IQL: $\tau \in (0, 1)$, the expectile of a random variable, and $\beta \in [0, \infty)$, an inverse temperature term for AWR. We use $\tau = 0.9$ and $\beta = 10.0$ and following [160], we use a cosine schedule for the actor

learning rate. For each task, we train an oracle goal-conditioned PPO agent for data collection. The agent’s initial state is randomly initialized. We collect an offline dataset of 1M environment transitions, roughly 2500 trajectories. We train IQL for 25000 offline gradient steps and report the average episode return across 5 random seeds.

E.5.1 Compute Resources and Runtime

All experiments can be run on a single Nvidia RTX A6000 GPU. Implementation is written completely in JAX. The following are rough estimates of average run-time for DynaMITE-RL and each baseline method for the online RL experiments with the HalfCheetah and ScratchItch environment. These numbers vary depending on the environment; JAX-based environments (e.g. Reacher and HalfCheetah) are highly parallelized and the runtimes are orders of magnitude lower than ScratchItch. We also run multiple experiments on the same device so runtimes may be over-estimated.

- RL²: 4 hour, 16 hours
- VariBAD: 3 hours, 8 hours
- BOREl: 3 hours, 8 hours
- SecBAD: 3 hours, 10 hours
- ContraBAR: 2.5 hours, 7 hours
- DynaMITE-RL: 3 hour, 8 hours

Appendix F

Robometer

F.1 Dataset Details

This section provides a detailed breakdown of the datasets used to train Robometer, including expert demonstration datasets, paired human-robot datasets, mixed-expertise datasets, and human-only datasets. We also describe the dataset filtering and task end-state adjustment procedures, and list the individual evaluation datasets used to measure generalization.

Paper	# Trajs.
RoboReward [175]	45k
RoboDopamine [274]	100k
VLAC [329]	300k
RBM-1M (ours)	1M

Table F.1: Comparison of known approximate trajectory counts across recent general-purpose reward modeling papers.

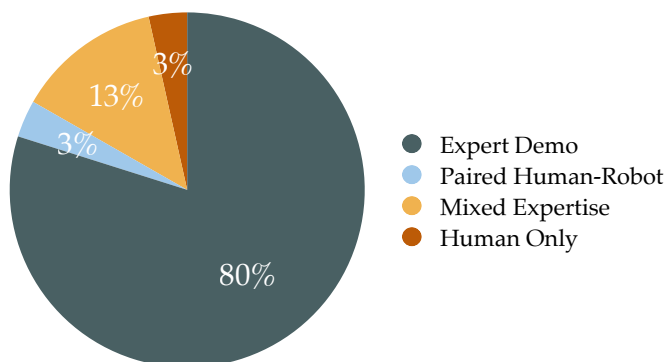


Figure F.1: Pie chart of RBM-1M dataset types. Full table with individual dataset details in Table F.4.

Table F.2: RBM-EVAL-ID - In distribution evaluation datasets overview. # Trajs. is computed by aggregating all evaluation splits per dataset, counting unique annotated trajectories.

Dataset	Embodiment	# Trajs.	Citation
OXE Eval Suite	Franka, Google Robot, Jaco 2, WidowX	14,398	[61]
RACER (Eval)	Franka Panda (simulation)	7,227	[66]
Metaworld Eval	Sawyer	151	[324]
LIBERO Eval	Franka Panda (simulation)	3,950	–
Total	5 unique robot embodiments	25,726	–

Table F.3: RBM-EVAL-OOD - Out-of-distribution evaluation datasets overview. # Trajs. is computed by aggregating all evaluation splits per dataset, counting unique annotated trajectories.

Dataset	Embodiment	# Trajs.
USC Franka	Franka Panda	24
USC Koch	Bimanual Kochv1.1	150
USC Trossen	Trossen	27
USC xArm	xArm	36
MIT Franka	Franka Panda	304
UT SO101	SO101	30
Total	5 unique robot embodiments	571

Table F.4: RBM-1M Training datasets overview. # Trajs. is determined by counting every unique video-language annotation (possibly across multiple views when available).

Dataset	Embodiment	# Trajs.	Citation
Expert Demonstration Datasets			
OXE Mix	11 mixed embodiments	449,475	[61]
AGIBotWorld-Alpha Subset	AgiBot G1 Bimanual Mobile Manipulator	216,911	[62]
Galaxea Open World	Galaxea R1 Lite Bimanual Mobile Manipulator	108,118	[142]
RoboReward OXE Mix	13 mixed embodiments	45,072	[175]
MolmoACT	Franka Panda	15,546	[173]
Humanoid Everyday	Unitree G1 & H1 Bimanual Humanoids	9,208	[341]
LIBERO	Franka Panda (simulation)	1,709	[197]
MetaWorld (ReWiND)	Sawyer (simulation)	100	[324, 333]
Paired Human–Robot Datasets			
MotIF	Stretch Mobile Manipulator	83	[130]
RH-20T	4 embodiments	29,969	[83]
PH2D	H1 (partly in MuJoCo simulation)	3,596	[241]
H2R	XArm	2,254	[306]
Mixed Expertise Datasets			
RoboArena Pairwise Comparisons	Franka Panda	12,379	[10]
SOAR Paired Success/Fail	WidowX	16,812	[343]
FAILSafe	Franka Panda (ManiSkill [276] simulation)	71,614	[195]
RACER Failure Dataset	Franka Panda (RLBench [138] simulation)	29,115	[66]
AutoEval Failed Trajectories	WidowX250	8,677	[344]
LIBERO Failure Dataset	Franka Panda (simulation)	1,473	[197]
Fino-Net Paired Failure (Baxter)	Baxter	229	[132]
Human-Only Datasets			
Epic-Kitchens	Human	37,030	[67]
Total	21 unique robot embodiments	1,059,370	–

F.1.1 Individual RBM-1M Training Dataset Details

Here, we list dataset curation details for every source in RBM-1M.

Preference Only Data. Some data sources vary significantly in when the teleoperator marks the end of the trajectory. As such, using these data sources for progress prediction and end-state success prediction negatively affects prediction accuracy. To avoid this issue, progress-only generalist robotic reward models would have to either manually label end states or simply forgo using these data sources. However, due to Robometer’s trajectory comparison-based preference prediction objective, we can still use these noisier datasets for preference prediction; we list which sources are used only for preference in each subsection.

Image Resolution and Frame Downsampling. We first perform data pre-processing before using the data for training. For storage efficiency, we downsample all trajectories to a maximum of 32 frames. Furthermore, we downsize image resolution such that the shortest edge (either height or width) of each image is 240 pixels. This procedure allows us to maintain the aspect ratio while ensuring the dataset does not consume excessive storage space. In total, the dataset size is around 6 TB.

Task End Thresholds. For predicting both progress and success ([Section 8.3.3](#)), it’s important that the end-frame of the trajectory corresponds to the timestep at which the task is actually finished. However, in most real-world datasets, this is not the case due to human teleoperator delay or different notions of when tasks are finished. We address this issue by manually setting this “task finished” threshold for each data source, which takes about 2 minutes per data source. See [Section F.1.2](#) for further details.

Expert Demonstration Datasets

We start with expert demonstration datasets, i.e., datasets that contain only successful trajectories demonstrated by competent teleoperators. For all of these datasets, the progress target we use in [Section 8.3.3](#) is 1.0.

Open-X Embodiment Mix We select a subset of datasets from the Open-X Embodiment (OXE) Dataset [61]. This subset was chosen from those selected for reward learning from prior work [333]. The subset includes DROID [149], Fractal [41], BRIDGE-v2 [292], Language Table [208], BC-Z [139], RoboSet [29], FurnitureBench [117], UT Austin Mutex [258], Berkeley Cable Routing [205], CLVR Jaco Play [70], Berkeley RPT [244], Toto [342], CMU Franka Pick-Insert [255], Stanford Hydra [22], Berkeley MVP [243], Berkeley Fanuc Manipulation [345], Mobile ALOHA [94], Imperial College Sawyer Wrist Cam, UCSD Kitchen [312], Austin BUDS [346], DLR Edan [242, 290], UTokyo LSMO [234], and NYU Rot [108].

We use wrist camera and external camera viewpoints from DROID due to the wide-angle of the DROID wrist cameras. For most other datasets, we only use external cameras unless wrist is the only viewpoint available.

Some of these datasets have validation or test-set splits; we use these splits for RBM-EVAL-ID.

For BC-Z and DLR Edan, we use these datasets only for preference prediction due to highly varied trajectory termination times relative to when the task was actually completed.

AGIBot World The AgiBotWorld-Alpha dataset [62] consists of 100k+ long-horizon trajectories on the AgiBot G1 bimanual mobile manipulator. We randomly select a 34,098 trajectory subset of the 100k trajectories to form our dataset. Each long-horizon trajectory is annotated with each shorter-horizon subskill that is required to accomplish the task. In total, this leads to 216,911 long-horizon and short-horizon trajectories. These short and long horizon skills include higher-level tasks such as “put all the oranges in the basket” and the lower-level skills for each “pick orange” and “place orange.” Although the dataset includes wrist cameras and multiple fisheye camera angles, we only include the egocentric head camera in our dataset.

Galaxea Open-World The Galaxea Open-World Dataset [142] is a large-scale humanoid dataset with 108,118 trajectories across 150 task categories. The dataset includes diverse tasks from pick-and-place to whole-body manipulation on a Galaxea R1 Lite bimanual mobile manipulator. Due to highly varied trajectory termination times relative to when the task was actually completed,

we use this dataset only for preference prediction.

RoboReward OXE + Roboarena Mix In addition to our own OXE mix, we incorporate the 45k trajectory OXE + Roboarena [10] training subset from RoboReward. This dataset is labeled using RoboReward’s VLM-based counterfactual instruction labeling technique to generate pseudo-failure instructions for successful trajectories. This technique complements Robometer’s training objectives — by directly incorporating all data into Robometer’s training mix and objectives. Final rewards for each trajectory are discrete numbers ranging from 1 to 5. We normalize their rewards to the range of $[0, 1]$, making this data suitable for all of our prediction objectives. We refer readers to Lee et al. [175] for further details. Similar to the OXE dataset mix, the DLR Edan and BC-Z subsets of the RoboReward data are also only used for preference prediction due to highly varied trajectory termination times relative to when the task was actually completed.

MolmoACT We use all external-view data from MolmoACT [173], collected on a Franka Panda arm, excluding trajectories with corrupted videos. We do not use wrist-cam data from MolmoACT because the camera angle is too narrow and often does not show the object being manipulated. We selected this dataset because it includes a diverse set of trajectories collected in clutter with good camera visibility.

Humanoid Everyday We use all data from [341], which contains Unitree G1 & H1 bimanual humanoid data from an egocentric viewpoint. This data was selected for diversity as it includes bimanual mobile data.

LIBERO LIBERO [197] provides a diverse set of simulated household manipulation tasks across 5 task suites. We use LIBERO- $\{10, \text{Object}, \text{Spatial}, \text{Goal}\}$ for RBM-1M and LIBERO-90 for evaluation in RBM-EVAL-ID. We follow OpenVLA’s [155] dataset re-generation scheme by re-rendering at 256x256, removing no-ops, and removing demonstrations which are not successful upon replay. We selected LIBERO for its use as a popular VLA benchmark and for the ease with

which we can generate our own failed trajectories across a diverse set of tasks. Thus, we also include a corresponding set of failed trajectories for all 5 task suites, constructed by replaying demonstration trajectories with added Gaussian noise to each action, mimicking policy execution error that results in failure.

Metaworld MetaWorld [324] is a multi-task simulated manipulation benchmark with a Sawyer arm. We use the 20-task training split consisting of 5 demonstrations each from Zhang et al. [333] for RBM-1M. Correspondingly, we use the 17-task evaluation dataset from Zhang et al. [333] for RBM-EVAL-ID. MetaWorld was selected early on for basic testing and ensuring that we can reproduce the results from ReWiND [333] in our own implementation of it. We kept it in RBM-1M for visual feature diversity.

Paired Human-Robot Datasets

MotIF A human-robot paired dataset with a Stretch Mobile Manipulator containing tasks involving motion-counting such as “stir 3 times” or shaking boba tea [130]. This dataset was selected for these counting tasks to encourage learning to track repetitive motions.

RH-20T A paired human-robot dataset for tabletop manipulation spanning four robot embodiments including Flexiv, UR5, Franka Panda, and Kuka robots [83]. This dataset was selected for its diversity in tasks and embodiments. The RH20T dataset consists of 7 configurations each with its own robot, table and camera setup. For each configuration, we select 1–2 camera views which both 1) capture the full scene and robot motion and 2) are consistent with the language instruction in terms of spatial relationships, e.g., left, right, top, bottom, etc. We removed null robot demonstrations without any arm movements, as well as demonstrations which seem to be the concatenation of demonstrations for multiple tasks.

PH2D A human-robot, real and simulation dataset containing Unitree H1 trajectories collected from an egocentric viewpoint [241]. This dataset was selected because it pairs simulation, real,

and human trajectories; it contains many pouring tasks and many very detailed task descriptions.

H2R A human-robot paired dataset with a UFACTORY XARM robot [306]. It contains pick-and-place and pushing tasks. We selected it because many tasks have multiple objects in the scene (e.g., a lighter tray and a darker tray to place a cube on), helping the reward model learn to better distinguish *correctly* manipulated objects.

Mixed Expertise Datasets

The data here contains paired successful and failed trajectories. We incorporate these mixed expertise data to encourage Robometer to effectively reward failed trajectories, which can help in a variety of domains (e.g., all downstream experiments in [Section 8.4.3](#)).

RoboArena Roboarena [10] data is from a set of human-performed evaluations of various generalist VLA policies on the DROID setup with a Franka Panda arm. Each evaluation has a partial progress score $\in [0, 1]$ with the vast majority of evaluations being failures. We use these progress scores solely to construct trajectory comparisons to predict over as trajectory termination times are highly varied and can essentially undo progress made in the middle of the trajectory, for which the human gave a partial progress score. We save videos from all available camera viewpoints, including wrist camera.

SOAR SOAR [343] data comes from autonomous policy rollouts guided by a VLM on a WidowX250 robot. Success/fail labels, generated by a VLM, are also provided. Due to automatic task generation and success/fail labeling, the dataset labels are quite noisy, and many trajectories contain tasks that are not possible in the scene, tasks that have already started from the first frame, and incorrect success/fail classifications. Therefore, we use a pre-trained Qwen-3-VL-4B [14] to filter out incorrectly labeled samples, including ones that were infeasible or unrelated to the task description. We do this by using the first, middle, and last frame to establish a general flow of the trajectory and ask the model to critique the positioning of relevant items. This is separated into a

stage prompt that empirically improved filtering quality using a small, manually verified set and ultimately filters out 45% of trajectories from the original dataset.

We save all filtered successful trajectories, and we also save all failed trajectories that have the same language description as at least one successful trajectory. Because this data does not contain progress labels and because trajectory end frames are highly variable with respect to when the task was actually completed in successful trajectories, we use the entire dataset only for generating trajectory comparisons for preference prediction.

FAILSafe FAILSafe [195] contains successful and failed trajectories from a Franka Panda collected in the Maniskill simulator [276]. Each task has many example failures collected from both wrist and external cameras. Tasks are also segmented into sub-tasks, e.g., reaching a cube → grasping the cube → ...

RACER RACER contains paired failed and successful trajectories on a Franka Panda in the RLBench simulator [66, 138]. This dataset was picked for its non-prehensile tasks, such as opening/closing drawers and sweeping.

AutoEval AutoEval contains data from automatic policy evaluations collected on 2 different WidowX250 setups [344]. While task diversity is limited, we used this dataset because it contains diverse strategies coming from the evaluation of arbitrary policies.

This dataset is used only for preference prediction due to some noisy automatic success/fail detection.

LIBERO We self-generated a failure dataset in LIBERO [197] by adding Gaussian noise to successful demonstration trajectories to re-generate paired failure trajectories for every task. This data is paired with the original success-only LIBERO dataset mentioned in [Section F.1.1](#).

Fino-Net Fino-Net data contains egocentric, paired success/fail data from a Baxter robot [132]. The data consists mainly of pick-and-place tasks, but it was selected for its use of a unique robot

not present in the other datasets.

Human only Datasets

Finally, we also include human-only data. Our final dataset contains just one dataset, Epic-Kitchens, but early on we also experimented with EgoDex [121]. We found that training solely on Epic-Kitchens helped predict rewards for robot data, but this was not the case with EgoDex, perhaps due to Epic-Kitchens’ background scene diversity and clutter.

Epic-Kitchens We include a subset of data from Epic-Kitchens [67]. Due to some difficulties we encountered in downloading the entire dataset, we picked a subset of Epic-Kitchens 100 uploaded to HuggingFace Datasets.¹

Furthermore, because language annotations and trajectory end times are quite noisy, we use the Epic-Kitchens dataset only for preference prediction.

F.1.2 Dataset Filtering and Task End-State Adjustment

To determine task-finished thresholds for each dataset, we designed a lightweight UI to visualize randomly sampled trajectories from each data source. With this visualizer, we sample 10 trajectories from each data source and manually mark the frame at which we deem the task to be complete. We define this threshold as the point at which the task description is satisfied. Then, we use the 90th percentile of end-frame thresholds (i.e., when 90% of the visualized trajectories are complete) as our threshold. Most teleoperators collecting data in RBM-1M define the trajectory as complete when the robot has performed a partial or full reset to neutral. As such, our end-state thresholds are typically set to around 80-95% of the trajectory length.

This process takes no more than 2 minutes per data source, and the thresholds are used to appropriately adjust target progress and success thresholds for training, detailed further in [Section F.2.2](#).

¹https://huggingface.co/datasets/awsaf49/epic_kitchens_100

F.1.3 Individual Evaluation Dataset Details

We summarize the number of trajectories in each of our in-distribution evaluation dataset in [Table F.2](#) and the out-of-distribution evaluation dataset in [Table F.3](#) and describe them in detail below.

USC Franka USC Franka is a dataset of mixed expertise trajectories collected between four different tabletop tasks such as “fold towel” and “put the plate on the sink.” For each quality label, we collected at least two trajectories each.

USC Trossen USC Trossen comprises mixed expertise trajectories collected between five different tabletop tasks using a bimanual Trossen. Some tasks are more articulated and dexterous such as “unzip the pencil case” and “stir the pot”. For each quality label, we collected at least two trajectories each.

USC Koch Arms USC Koch Arms replicates the real-world data collected in ReWiND [\[333\]](#) using bimanual Koch Arms. This dataset consists of 10 demos per-task over 20 tasks. We also collect suboptimal and failure examples for each task.

MIT Franka MIT Franka is a dataset composed of diverse tasks, including pick and place (“pick up the banana from X and place it on Y”) and dexterous tasks such as “fold the towel in half”, “pick up the spatula and stir the beans in the pot”. We also include a task “pick up the mouse and place it on X while avoiding Y” that requires semantic scene understanding. We collect trajectories of different levels of expertise for each task.

UT SO-101 Univ3 SO-101 is a real-world dataset of manipulation trajectories collected using a single-arm SO-101 robot. The dataset comprises two mixed-quality settings, each containing both successful and failure trajectories. The first setting is a clean pick-and-place environment centered on a single household task, “put the bread in the oven.” This setting includes 30 successful

demonstrations and 45 failure trajectories. The second setting is a cluttered multitask environment consisting of three pick-and-place tasks, such as “put the red bowl on the blue plate.” For each task in this setting, we collect 20 successful demonstrations and 15 failure trajectories.

F.2 Model Details

This section describes the Robometer architecture and training setup in detail, including model hyperparameters, training objectives with associated loss weights, and the computational resources used for training.

F.2.1 Model Architecture and Training Parameters

We summarize the training configuration for Robometer in [Table F.5](#).

Table F.5: Training Configuration for Robometer

Parameter	Value
Base Model	Qwen/Qwen3-VL-4B-Instruct
Number of frames	8
Per-device batch size	16
Learning rate	2×10^{-5}
Weight decay	0.01
Total training steps	6500
Max sequence length	1024
LR scheduler	Cosine
Warmup ratio	0.1
Min frames per trajectory	5
Progress loss type	Discrete
Number of discrete bins	10
MLP head num hidden layers	1
MLP head dropout	0.1
MLP head hidden dim	2048

Architecture We illustrate the overall architecture of Robometer in Figure F.2. Robometer instantiates a causally masked vision–language model (VLM) backbone, QWEN3-VL-4B-INSTRUCT, a unified transformer that processes interleaved text and visual tokens using a single autoregressive decoder. Natural language instructions are tokenized using Qwen’s SentencePiece-based tokenizer.

Each video trajectory is first subsampled into a fixed number of frames. Each frame is independently encoded by a ViT-style visual encoder and projected into a sequence of visual tokens. A special $\langle | \text{video_start} | \rangle$ token marks the beginning of visual input, after which visual tokens are appended sequentially and assigned unique positional indices in the unified token sequence. The decoder jointly attends over language tokens, visual tokens, and special tokens using a single causal attention mask, supporting unified multimodal reasoning.

To enable dense, frame-level reward estimation, a learned progress token $\langle | \text{prog_token} | \rangle$ is inserted after each frame o_t^1 in the first trajectory. Under causal masking, the hidden state of $\langle | \text{prog_token} | \rangle$ can attend only to the instruction and visual tokens from frames $o_{1:t}^1$.

We introduce a dedicated separator token $\langle | \text{split_token} | \rangle$ to demarcate the boundary between two video trajectories. The second trajectory o^2 is appended after $\langle | \text{split_token} | \rangle$ and processed jointly with o^1 in the same causal sequence. A learned preference token $\langle | \text{pref_token} | \rangle$ is appended at the end of the prompt; its hidden state aggregates information from the instruction and both trajectories and is used to predict pairwise preference.

We attach lightweight MLP heads to the shared VLM backbone. Specifically, the progress, preference, and success heads each consist of a two-layer MLP followed by LayerNorm, GELU activation, and dropout, and a final linear projection that outputs a scalar logit. The progress head is applied to the hidden states of the interleaved $\langle | \text{prog_token} | \rangle$ tokens to produce frame-level progress logits, while the success head operates on the corresponding per-frame hidden states to predict frame-level success logits. The preference head is applied to the hidden state of $\langle | \text{pref_token} | \rangle$ to produce a single logit indicating whether the first trajectory is preferred over the second.

For preference supervision, we construct a single multimodal prompt that contains two trajectories, serialized into a single causal sequence and separated by a split token. This is a more detailed, expanded version of Equation (8.3.1).

$$\begin{aligned} \text{Tok}(l, o^A, o^B) \rightarrow & \text{Tok}(l) \text{ This is Trajectory A.} \langle | \text{video_start} | \rangle \text{Tok}(o_{1:T}^A) \\ & \langle | \text{split_token} | \rangle \text{ This is Trajectory B.} \\ & \langle | \text{video_start} | \rangle \text{Tok}(o_{1:T}^B) \langle | \text{pref_token} | \rangle. \end{aligned}$$

Prompt. Since we train the model on all 3 objectives (progress, success, preference) simultaneously, we always sample a preference prompt. Thus, we always condition the model on the following natural-language prompt.

Robometer training prompt

Given these two trajectories for the task “{task}”, evaluate which one makes more progress towards the task. Return A for the first trajectory and B for the second trajectory. Additionally, predict the task progress at each frame of the first trajectory as a float between 0 and 1, where 0 corresponds to the initial state and 1 corresponds to task completion. If the robot is not performing the specified task, predict 0 progress.

Model and Training Params We list overall hyperparameters in Table F.5. We did not extensively sweep these hyperparameters—we followed best practices and parameters from prior work [14, 15, 175]. For the preference, success, and progress prediction MLPs, we heuristically select a hidden dimension of 2048, which is half the input size (Qwen’s hidden embedding size) of 4096.

F.2.2 Training Objectives

Trajectory Cutoffs and Success Supervision In several teleoperated datasets, episode termination does not coincide with task completion. Operators typically complete the task and then manually stop the recording after a short delay, resulting in trailing frames in which the robot remains static or performs incidental motions unrelated to task execution. These frames do not

reflect additional task progress and introduce noise when used for frame-level progress supervision. To mitigate this issue, we manually annotate a dataset-specific success cutoff corresponding to the frame at which the task objective is first achieved and apply this cutoff uniformly to all trajectories within the dataset (see Table F.6).

For frames occurring after the dataset-specific success cutoff, we assign target progress and success labels of 1.0, reflecting that the task has already been completed. Success supervision is applied selectively to avoid ambiguous or conflicting learning signals. Specifically, we predict success only for frames whose target progress is either strictly below a minimum success threshold τ_{succ} , corresponding to clearly pre-completion states, or exactly equal to 1.0, corresponding to completed states. Frames with intermediate progress values near completion are excluded from success supervision, as they often correspond to visually ambiguous transitional or stabilization phases.

Progress Supervision in Preference Samples For preference samples, we apply progress supervision only to the first trajectory (Trajectory A), reflecting the fact that progress prediction is used at inference time for a single video. To avoid introducing noisy or ill-defined targets, we compute progress loss for Trajectory A only when it corresponds to a successful trajectory. When Trajectory A corresponds to a failed or suboptimal execution, we do not apply progress supervision.

For datasets that provide continuous partial completion annotations, such as RoboArena, we supervise progress on the final frame of Trajectory A using the ground-truth `partial_success` value, even when the trajectory is suboptimal. This allows the model to learn calibrated progress estimates from human-annotated partial completion labels.

Data Source and Strategy Sampling Detailing Section 8.3.4 further, we construct pairs of trajectory comparisons for preference supervision (and also progress/success supervision with the first trajectory input to the model) by first sampling a *preference sampling strategy* from *different-*

Table F.6: Dataset-specific success cutoffs used to correct for delayed episode termination in teleoperated data. Datasets not contained here use a default success cutoff of 1.0.

Dataset	Success Cutoff
<i>Open-X Embodiment (OXE)</i>	
Aloha Mobile	0.95
Austin BUDS	0.95
Berkeley Cable Routing	0.95
Berkeley FANUC Manipulation	0.98
Bridgev2	0.95
DLR EDAN Shared Control	0.95
CMU Pickup Insert	0.95
UCSD Kitchen	0.95
UT Austin Mutex	0.95
BC-Z	0.95
Berkeley MVP	1.00
Berkeley RPT	0.76
Fractal	1.00
Furniture Bench	1.00
DROID	0.95
Imperial College Sawyer Wrist Cam	0.90
Language Table	1.00
NYU ROT	0.70
RoboSet	0.85
Stanford HYDRA	1.00
Tokyo-U LSMO	1.00
TOTO	1.00
<i>Other Datasets</i>	
MolmoAct Household	0.94
MolmoAct Tabletop	0.94
AgibotWorld	0.95
PH2D	0.95
RH20T (Human)	0.92
RH20T (Robot)	0.94
RoboArena	0.90
H2R	0.90
SOAR	0.95
AutoEval	0.94
Galaxea	0.80
FINO-Net	0.75
Humanoid Everyday	0.80
Motif	0.95

task, *rewind augmentation*, or *different-expertise*. Conditioned on the selected strategy, we restrict sampling to datasets from which the corresponding type of preference pair can be constructed.

- For *different-expertise* strategy, samples are constructed from mixed expertise datasets, such as RACER or FINO-Net, where trajectories are labeled as successful, suboptimal, or failed. We use these annotations to form preference pairs that rank trajectories by execution quality. In datasets such as RoboArena that additionally provide continuous `partial_success` annotations, we sample two trajectories from the same task and assign the trajectory with higher partial success as the preferred one.
- *Rewind augmentation* can be applied to trajectories from any dataset and does not require additional annotations.
- For *different-task pairing*, we sample a trajectory that is successful for one task and pair it with a trajectory executing a different task. When constructing different-task pairs, we control whether both trajectories are drawn from the same data source or from different sources. With probability ρ_{same} , we sample different-task pairs from the same dataset to discourage reliance on dataset-specific visual cues, while with probability $1 - \rho_{\text{same}}$ the trajectories are drawn from different datasets to encourage robustness to domain shift. We set $\rho_{\text{same}} = 0.5$, yielding an equal mix of same-source and cross-source different-task comparisons.

Preference Prediction Loss: Bradley-Terry vs BCE Our preference prediction loss in Equation (8.3.2) uses a binary cross-entropy loss coming from an MLP on top of the $\langle | \text{pref_token} | \rangle$ embedding in Equation (8.3.1). Using this token with *two* input videos allows Robometer to simultaneously attend to *both* videos to predict this loss, resulting in one forward and backward pass to train Robometer on preference prediction.

A common alternative in both reward modeling from pseudo-preferences [315] and general RLHF reward function training on real human preferences [56] is to instead use the Bradley-Terry loss [38, 56], where a single preference score is computed over an entire individual trajectory at

a time, and then the loss is backpropagated across batch comparisons.

Ignoring computational efficiency differences, our main reason for not implementing the Bradley–Terry loss into Robometer is that it does not leverage the pre-trained attention mechanism as effectively: our two-video formulation allows tokens from one trajectory to explicitly attend to tokens from the other, enabling direct cross-video comparison during preference prediction. This strategy is common for comparative reasoning tasks in language reference games [19, 217, 218]. In contrast, Bradley–Terry computes independent scalar scores per trajectory and only couples them through the loss, which can make learning more sensitive to score calibration across batches. We compare these objectives in Section F.4. As shown in Table F.7, predicting a preference label from a dedicated $\langle \text{pref_token} \rangle$ given both videos (BCE) improves reward alignment (from 0.862 to 0.948) and trajectory ranking over Bradley–Terry (from 0.325 to 0.655).

Table F.7: Bradley–Terry (BT) versus preference label from a dedicated $\langle \text{pref_token} \rangle$ (BCE) given both videos in a single forward pass, evaluated on RBM-EVAL-00D.

Pref. Loss	VOC $r \uparrow$	Kendall $\tau \uparrow$	Succ–Fail Diff. $\uparrow$
BT	0.862	0.325	0.242
BCE	0.945	0.643	0.320

F.2.3 Computational Resources

We train Robometer using a per-GPU batch size of 16 across 4 GPUs, resulting in an effective batch size of 64. All experiments are run on a server with jobs requesting 4 NVIDIA H200 GPUs and 32 CPU threads for 6.5k training steps, corresponding to approximately 2 days of wall-clock training time. Unless otherwise specified, we use the same training configuration across all experiments.

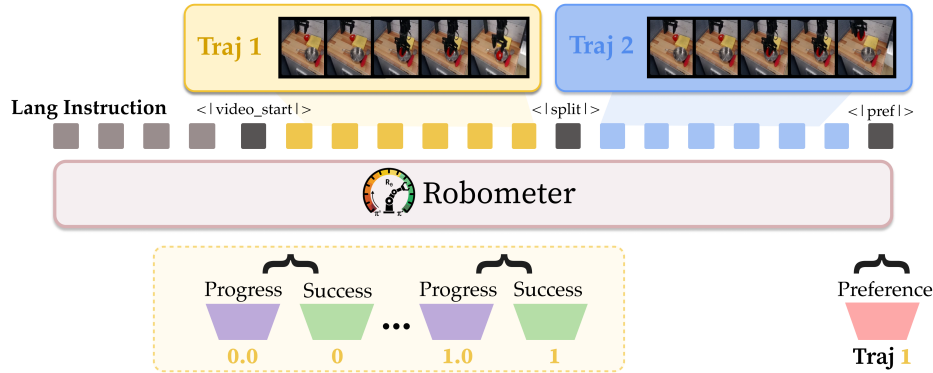


Figure F.2: Robometer model architecture. A VLM backbone processes a language instruction with one or two trajectories. For both single and paired inputs, per-frame hidden states of the first trajectory are passed to MLP heads predicting dense task progress and binary success. For paired trajectories, a preference token is used with a preference head to select the trajectory that better satisfies the instruction. The model is trained with joint progress, success, and preference objectives.

F.3 Additional Reward Evaluation Results

We first list baseline implementation details for main paper baselines before discussing additional reward evaluation results.

VLAC VLAC released 2 models built on the InternVL [52] VLM, with 2B and 8B parameters. VLAC takes 2 frames as input and predicts the relative increase in progress between the first and second frames; thus, its output range is $[-1, 1]$. We directly perform inference using the publicly available GitHub code and test both the 2B and 8B models—we found the 8B model to perform slightly better on our evaluation results and thus use VLAC-8B as our baseline. The pretrained VLAC checkpoints are obtained from the authors’ public release.² For relevant reward comparisons, we normalize its outputs to make them directly comparable to those of all other models that output only positive rewards.

Robo-Dopamine trains a step-aware *general process reward model* (GRM) to estimate fine-grained manipulation progress from *multi-view* observations by predicting discretized progress “hops” between states and aggregating these signals into a dense reward. We use the authors’

²<https://huggingface.co/InternRobotics/VLAC>

released inference code and 8B checkpoint.³⁴ Robo-Dopamine relies on richer task-specific context at inference time (e.g. such as multi-view and goal images), which may not be consistently available across datasets or deployment settings.

GVL We compare against GVL, which queries a closed-source vision-language model to estimate task progress directly from video frames. GVL does not involve model fine-tuning; instead, progress is inferred solely through prompting at inference time. In our experiments, we use GPT-5-mini,⁵ which is the strongest-performing closed-source model reported in the RoboRewardBench [175] benchmark. To discourage the model from exploiting trivial temporal cues and to reduce correlations with frame index, GVL shuffles video frames during inference and prompts the model to predict progress values that are subsequently reordered chronologically.

TOPReward prompts a pre-trained VLM with the question “does the trajectory complete the task?” for a given task instruction and robot trajectory. Then, it extracts the log-likelihood of the “true” token as the reward before normalization to bound the reward between 0 and 1. They experiment on: Gemini-2.5-Pro [277], Molmo2 [57], and Qwen-3-VL-Instruct-8B [14], finding that Qwen-3 works the best overall on VOC scores. Thus, we use TOPReward with Qwen-3-VL-8B in our experiments and for cross-trajectory comparisons (such as Kendall- τ_a) we do not normalize intra-trajectory rewards (otherwise most trajectories will end at near-1.0 reward).

ReWiND We implement ReWiND [333], which operates on precomputed visual and language embeddings rather than raw pixels. Each video frame is encoded using a frozen DINOv2 vision encoder [233], and task instructions are embedded using the frozen Sentence-Transformers MiniLM-L6-v2 model (all-MiniLM-L6-v2 [247]). The resulting visual and text embeddings are linearly projected into a shared latent space and processed by a Transformer encoder.

To enable a direct architectural comparison with Robometer, we predict per-frame progress

³<https://github.com/FlagOpen/Robo-Dopamine>

⁴<https://huggingface.co/tanhuajie2001/Robo-Dopamine-GRM-2.0-8B-Preview>

⁵GPT-5-mini-2025-08-27 the latest version as of writing.

and success directly from the hidden states corresponding to visual frame embeddings using lightweight MLP heads, and train progress using a discrete bin formulation. In our ablation study in Table 8.3, we further extend the ReWiND baseline with a preference objective by concatenating two trajectories into a single sequence and predicting a binary preference label via a learned preference token, and we scale up its original architecture (4 layers, 8 attention heads) to a hidden dimension of 1024, 32 transformer layers, and 16 attention heads.

RoboReward is trained to predict a discrete 1-5 progress target on OXE and RoboArena. RoboReward is trained using trajectory-level supervision and does not model intermediate task progress within a trajectory. As a result, the model is primarily designed to provide a sparse, terminal reward. For fair evaluation and comparison with our dense reward formulation, we obtain frame-level rewards from RoboReward by running inference independently on each frame of a trajectory and treating the resulting predictions as per-frame rewards. We use the publicly released RoboReward-8B pretrained checkpoint provided by the authors.⁶⁷

F.3.1 Preference Prediction

RL-VLM-F RL-VLM-F [298] predicts trajectory preferences by prompting a closed-source vision-language model to compare the *final frame* of two trajectories conditioned on a task description. In our experiments, we instantiate RL-VLM-F using the same OpenAI GPT-5-mini model as for GVL. RL-VLM-F uses the following fixed prompt for preference prediction.

⁶<https://huggingface.co/teetone/RoboReward-4B>

⁷<https://huggingface.co/teetone/RoboReward-8B>

RL-VLM-F preference prompt

Each frame comes from a robot trajectory. (Think causally and use image comparison to verify any confusion between the base of the robot and the end effector.)

1. What is shown in the first image (Image A)?
2. What is shown in the second image (Image B)?
3. For this question, here is the Goal Text: GOAL_TEXT

Is the goal being better achieved in Image A or Image B?

Reply with a single line containing 0 if the goal is better achieved in Image A, or 1 if the goal is better achieved in Image B. Reply -1 if there is no discernible difference or progress.

For RL-VLM-F, preferences are inferred using only the final frame of each trajectory, whereas Robometer applies a learned preference head that directly compares full video sequences. Preference accuracy is computed as the fraction of pairwise comparisons in which the predicted ordering matches the ground truth. To evaluate preference quality, we construct trajectory pairs from the RBM-EVAL-00D split along two axes: (i) differing task instructions and (ii) differing trajectory quality labels. For each dataset, we randomly sample 500 pairwise trajectory comparisons and evaluate predicted preferences against ground-truth ordering labels. As shown in [Tables F.8](#) and [F.9](#), Robometer consistently outperforms RL-VLM-F, improving average preference accuracy by 27.0% on different-quality pairs and by 32.4% on different-task pairs.

Table F.8: RL-VLM-F vs Robometer on Trajectory Quality. Different quality trajectory pairwise preference accuracy on 500 comparisons for each of the RBM-EVAL-00D datasets.

Dataset	RL-VLM-F (%)	Robometer (%)
USC Franka	52.1	75.0
USC Koch	54.4	79.4
USC Trossen	66.7	76.2
USC xArm	48.6	88.9
MIT Franka	54.4	85.4
UT SO-101	56.7	90.0
Average	55.5	82.5

Table F.9: RL-VLM-F vs Robometer on Different Tasks. Different task trajectory pairwise preference accuracy on 500 comparisons for each of the RBM-EVAL-OOD datasets.

Dataset	RL-VLM-F (%)	Robometer (%)
USC Franka	70.7	100.0
USC Koch	54.7	89.8
USC Trossen	64.0	99.0
USC xArm	73.3	98.2
MIT Franka	55.3	98.4
UT SO-101	72.7	100.0
Average	65.1	97.6

Table F.10: Per-dataset model ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failed trajectories on RBM-EVAL-OOD

Dataset	VOC $r \uparrow$			Kendall $\tau \uparrow$			Succ-Fail Diff. $\uparrow$		
	Prog. Only	+Preference	Robometer	Prog. Only	+Preference	Robometer	Prog. Only	+Preference	Robometer
USC Franka	0.913	0.974	0.949	0.083	0.542	0.625	0.039	0.428	0.326
USC Koch Arm	0.933	0.932	0.967	0.231	0.357	0.477	0.081	0.142	0.191
USC Trossen	0.199	0.902	0.910	0.333	0.423	0.569	0.052	0.231	0.312
USC xArm	0.890	0.973	0.980	0.389	0.597	0.750	0.079	0.154	0.345
MIT Franka	0.936	0.942	0.945	0.183	0.458	0.572	0.063	0.223	0.310
UT SO101	0.964	0.899	0.916	0.533	0.667	0.867	0.134	0.244	0.438
Average	0.806	0.939	0.945	0.292	0.507	0.643	0.075	0.237	0.320

Table F.11: Per-dataset VOC reward alignment results. Pearson correlation results (VOC) across individual datasets for RBM-EVAL-ID and RBM-EVAL-OOD.

Split	Dataset	Baselines				w/ RoboReward Training Data			w/ our RBM-1M data	
		GVL	VLAC-8B	RoboDopamine-8B	TOPReward	RoboReward-4B	RoboReward-8B	Robometer	ReWiND	Robometer
RBM-EVAL-ID	RACER (Val)	0.131	0.156	0.752	0.239	0.491	0.652	0.937	0.561	0.950
	OXE (BC-Z Eval)	0.142	-0.150	0.767	0.413	0.643	0.809	0.683	0.442	0.785
	OXE (Berkeley Cable Routing Eval)	0.075	-0.425	0.728	0.151	0.657	0.801	0.900	0.491	0.915
	OXE (Bridge V2 Eval)	0.143	-0.875	0.830	0.427	0.891	0.898	0.633	0.526	0.878
	OXE (Jaco Play Eval)	0.114	-0.151	0.873	0.499	0.793	0.785	0.861	0.550	0.894
	OXE (Toto Eval)	0.254	-0.416	0.732	0.734	0.886	0.939	0.947	0.340	0.973
	OXE (Viola Eval)	0.264	0.479	0.750	0.463	0.915	0.896	0.967	-0.014	0.988
	Metaworld (Eval)	0.134	0.211	0.773	0.238	0.746	0.779	0.737	0.630	0.929
	Libero (90)	0.254	0.217	0.876	0.586	0.874	0.846	0.912	0.592	0.962
	Average	0.168	0.089	0.787	0.417	0.766	0.823	0.842	0.458	0.919
RBM-EVAL-ODD	USC Franka	0.102	0.356	0.842	0.480	0.909	0.909	0.959	0.772	0.949
	USC Koch	0.176	0.074	0.847	0.384	0.866	0.916	0.969	0.585	0.967
	USC Trossen	0.542	0.256	0.647	0.390	0.781	0.726	0.925	0.226	0.910
	USC xArm	0.282	0.454	0.840	0.160	0.896	0.914	0.951	0.435	0.980
	MIT Franka	0.268	0.601	0.733	0.471	0.896	0.899	0.868	0.531	0.945
	UT SO101	0.203	0.511	0.868	0.497	0.926	0.930	0.888	0.497	0.916
	Average	0.262	0.375	0.796	0.397	0.879	0.882	0.927	0.508	0.945

Table F.12: Per-dataset trajectory ranking results. Trajectory ranking results on individual RBM-EVAL-00D datasets.

Dataset	Baselines					w/ RoboReward Training Data			w/ our RBM-1M data	
	GVL	VLAC-2B	VLAC-8B	RoboDopamine-8B	TOPReward	RoboReward-4B	RoboReward-8B	Robometer	ReWiND	Robometer
USC Franka	0.250	0.292	0.271	0.479	0.167	0.625	0.625	0.583	-0.125	0.625
USC Koch	-0.008	0.167	0.064	0.442	0.183	0.332	0.264	0.533	0.336	0.477
USC Trossen	0.292	-0.111	-0.417	0.333	-0.292	0.333	0.389	0.646	0.028	0.569
USC xArm	0.056	0.167	0.139	0.431	-0.111	0.528	0.347	0.403	-0.167	0.750
MIT Franka	0.306	-0.017	0.072	0.333	0.286	0.494	0.396	0.479	0.080	0.572
UT SO101	0.300	-0.033	0.167	0.700	0.533	0.700	0.767	0.667	-0.067	0.867
Average	0.199	0.077	0.049	0.453	0.128	0.502	0.465	0.552	0.014	0.643

F.3.2 Robometer in Long Horizon Tasks

Robometer’s progress predictions are conditioned on the *instruction*, rather than simple geometric proximity; as long as the instruction is sufficiently detailed, it can assign increasing progress even when the robot temporarily moves away from intermediate objects to complete the task. We analyze this behavior in Figure F.3 on the “*Make coffee*” task, which consists of multiple steps such as picking up the pod, closing the lid, and placing the cup correctly.

When the task instruction is brief or underspecified, Robometer’s predicted progress tends to stagnate or occasionally decrease. In contrast, with more detailed instructions, the model assigns consistently increasing rewards as the task progresses, shown on the right side of the figure. An alternative approach to handling long-horizon tasks is to decompose them into multiple stages, as demonstrated in our multi-stage Automatic Online RL experiments (Section 8.4.3).

F.3.3 RoboRewardBench Evaluation

We further evaluate on the *external* RoboRewardBench benchmark [175], reporting Mean Absolute Error (MAE) on rewards discretized into 1–5 scores. Baseline results under the same protocol are shown in Table F.13 with a Robometer variant using only RoboReward data with matching 5-bin outputs. This RoboReward-only Robometer variant attains an MAE of 0.75, outperforming RoboReward-4B. Finally, note that in our OOD evaluations in Table 8.1, RoboReward-8B and 4B perform similarly, and GPT-5-mini with GVL performs markedly worse. We therefore attribute the stronger results of larger models on RoboRewardBench to its 5 discrete labels and its final-

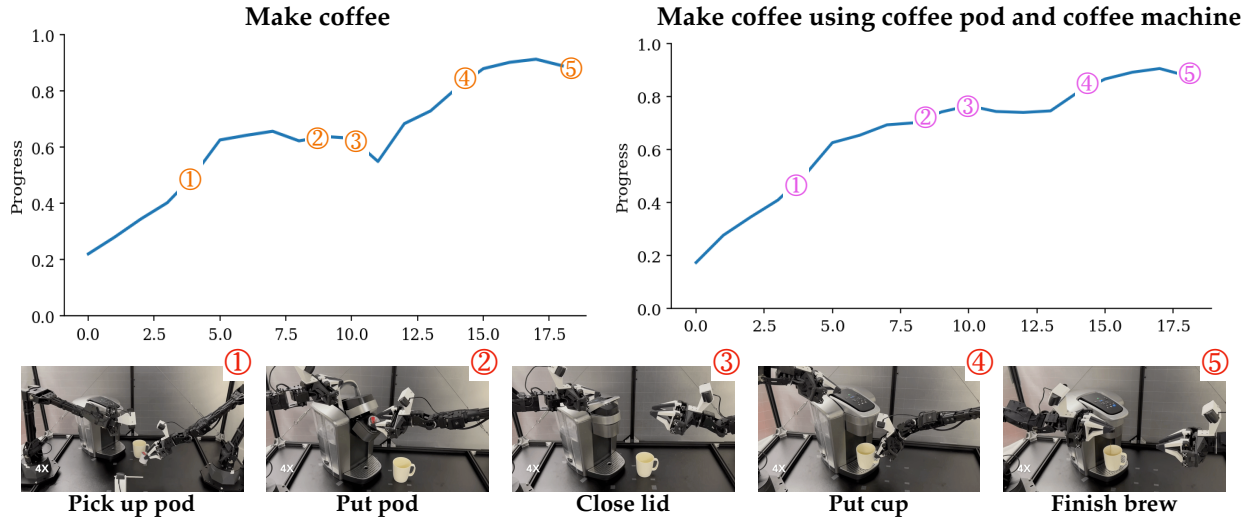


Figure F.3: Qualitative analysis of Robometer on a long-horizon task. With sufficiently detailed task instructions, Robometer correctly assigns increasing rewards across multiple stages of the task.

frame-only evaluation protocol.

Model Type	Model	RoboRewardBench MAE (↓)
	Robometer (only RoboReward data)	0.75
Qwen3-4B	RoboReward-4B	0.85
Models	Qwen3-VL-4B-Instr.	1.03
	RoboReward-8B	0.67
Closed /	GPT-5-mini	0.69
Larger	Qwen3-VL-8B-Instr.	0.89
	Gemini-2.5-pro	0.90

Table F.13: Evaluation on the RoboRewardBench benchmark [175].

Table F.14: Data sampling strategy ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failure trajectories on RBM-EVAL-00D and LIBERO-90. Models are trained without using RBM-EVAL-00D scenes; RBM-EVAL-00D and LIBERO-90 are held out for evaluation.

Model	VOC $r \uparrow$		Kendall $\tau \uparrow$		Succ-Fail Diff. $\uparrow$	
	RBM-EVAL-00D	LIBERO (90)	RBM-EVAL-00D	LIBERO (90)	RBM-EVAL-00D	LIBERO (90)
No Different Task	0.930	0.966	0.560	0.903	0.260	0.362
No Rewind	0.860	0.818	0.480	0.815	0.150	0.241
No Subopt	0.915	0.910	0.585	0.890	0.235	0.413
Ours	0.945	0.976	0.643	0.919	0.320	0.455

Table F.15: Loss weight ablation results. Hyperparameter sweep over preference/progress loss weights ($\lambda_{\text{pref}}, \lambda_{\text{prog}}$) on LIBERO-10 and LIBERO-90. Each model is trained on LIBERO-(10, Spatial, Object, Goal) and LIBERO-90 is heldout for evaluation only.

Weights ($\lambda_{\text{pref}}, \lambda_{\text{prog}}$)	VOC $r \uparrow$		Kendall $\tau \uparrow$		Succ-Fail Diff. $\uparrow$	
	LIBERO-10	LIBERO-90	LIBERO-10	LIBERO-90	LIBERO-10	LIBERO-90
(2, 1)	0.983	0.964	0.981	0.898	0.347	0.352
(1, 2)	0.986	0.946	0.982	0.875	0.423	0.367
(1, 1)	0.994	0.976	0.986	0.919	0.483	0.455

F.4 Additional Ablations

F.4.1 Preference pair sampling ablations

As shown in Table F.14, we ablate our preference pair sampling strategies to isolate which parts of the data construction are responsible for *generalization*. We evaluate on held-out RBM-EVAL-00D and the out-of-distribution LIBERO-90 benchmark, neither of which is used for training. We remove one strategy at a time from Section 8.3.4 to measure its contribution to held-out ranking and reward separation.

Across both held-out benchmarks, removing any component hurts performance, with the largest drop coming from disabling **trajectory rewinding**, which most strongly reduces success-failure separation and degrades trajectory ranking. **Different-task negatives** primarily affect instruction grounding: dropping them yields a consistent (though smaller) degradation in OOD

ranking. Finally, **suboptimal trajectory** pairs help calibrate rewards across mixed-quality behavior; removing them reduces ranking quality and narrows the margin between successful and failed trajectories. Overall, each preference construction contributes a complementary signal, and combining them yields the strongest held-out ranking and reward separation.

F.4.2 RBM-EVAL-OOD Pre-Training Objective Ablations

As shown in Table F.10, incorporating trajectory-level preference supervision consistently improves reward-model behavior compared to training on progress labels alone. On average, adding preferences boosts reward alignment improves trajectory ranking, indicating that pairwise comparisons provide a strong signal for ordering failed, suboptimal, and successful behaviors. Robometer achieves the best average Kendall score and the largest success–failure final reward difference, while maintaining high alignment. Compared to the *+Preference* variant, Robometer additionally incorporates failure trajectories during training, thereby further improving discrimination between successful and unsuccessful behaviors. Notably, Robometer yields consistent gains across all OOD datasets, suggesting that jointly leveraging dense progress targets, preference supervision, and failure data produces a reward function that both tracks task completion and better discriminates overall trajectory quality in OOD settings.

Moreover, Table F.16 shows that removing the success head leads to a small drop in reward generalization, indicating that success prediction provides a useful training signal beyond its deployment-time use.

Table F.16: Success prediction head ablation results. Reward alignment, trajectory ranking, and final reward difference between successful and failure trajectories on RBM-EVAL-OOD and LIBERO-90.

Model	LIBERO-90			RBM-EVAL-OOD		
	VOC $r \uparrow$	Kend. $\tau \uparrow$	S-F $\uparrow$	VOC $r \uparrow$	Kend. $\tau \uparrow$	S-F $\uparrow$
Robometer (-succ)	0.97	0.89	0.40	0.94	0.60	0.27
Robometer	0.98	0.92	0.46	0.94	0.64	0.32

Additionally, we report detailed evaluation results for each dataset in RBM-EVAL-OOD in Table F.11 (reward alignment) and Table F.12 (trajectory ranking) for all baselines.

F.4.3 Training loss weight ablations

Table F.15 sweeps the relative weighting between the preference and progress losses. We find that the best performance is achieved when the two objectives are weighted uniformly, with $(\lambda_{\text{pref}}, \lambda_{\text{prog}}) = (1, 1)$ outperforming settings that upweight either preference or progress on LIBERO-90 across alignment, ranking, and success–failure separation.

F.5 Policy Learning Experiment Details

F.5.1 RL with Ablated Reward Models

Experiment Setup. For each ablated reward model described in Section 8.4.2, we train an RL policy from scratch using SAC [105]. We evaluate performance on two tasks from the LIBERO-90 suite: Task 28 (*close the top drawer*) and Task 33 (*close the microwave*). Both reward estimation and RL training use only the external camera view. The policy observations consist of a DINO-v2-small [233]-featurized external image concatenated with proprioceptive state inputs. During training, we perform 25 evaluation episodes every 5000 training steps and report the average success rate $\pm$ standard deviation. For each reward model, we train 5 random seeds. The SAC hyperparameters are provided in Table F.17.

F.5.2 Automatic Online RL

Environment setup. We setup a DROID [149]-style Franka Panda environment on a cluttered tabletop as depicted in Figure 8.6. Following DROID, we have a Robotiq gripper, an exterior Zed camera mounted to the left of the robot arm, and a Zed wrist camera.

- **Single Stage:** The task is to put the bowl onto the table, where the difficulty lies in the bowl starting in a tall dish rack which physically blocks the robot gripper if it approaches the bowl from too low of an angle. The clutter also makes it difficult for π_0 to perform this task with high success rates zero-shot (20% initial success rate), making it well-suited for dense reward RL. The task instruction given to π_0 is “put the bowl on the table.”

Table F.17: RL with Ablated Reward Models. SAC hyperparameters.

Hyperparameter	Value
Batch size	128
Target update rate τ	0.005
Discount factor γ	0.99
Target update interval	1
Number of critics	5
Critics sampled per update	2
Pooled critic features	True
Actor updates per train step	1
Critic updates per actor update	1
Actor learning rate	1×10^{-5}
Critic learning rate	1×10^{-5}
Entropy coeff learning rate	3×10^{-4}
Target entropy	0
Learning starts	5000 steps

- **Multi-Stage:** The first task is to put the corn in the pot, and the second is to then put the lid on the corn, emulating a “steam corn” task. The pot being in the dish rack confuses π_0 , so the base policy tends to either miss putting the corn in the pot or collide with the dish rack and get stuck. The exact instructions for π_0 are “put the corn in the pot located in the dish rack” and “put the lid on the pot located in the dish rack.”

RL algorithm. We train an RL policy from scratch with Diffusion Steering (DSRL) [291] to steer a frozen π_0 policy pre-trained on the DROID [149] dataset. The DSRL policy trains an SAC [105]-style algorithm that operates over the *noise space* of the π_0 flow matching head. Specifically, the inputs to the RL policy are:

- DINO-v2-small [233]-featurized wrist image (384-dim)
- π_0 VLM hidden embedding (2048-dim)
- Proprioceptive joint and gripper positions (8-dim)
- For multi-stage only: the index of the current stage the policy is in (1-dim), concatenated with the proprioception.

The output is an action chunk of length 8, where each action is of dimension 32 (π_0 's flow matching head noise dimension) and the action bounds are $[-2.0, 2.0]$ for single-stage and $[-1.5, 1.5]$ for multi-stage. The original DSRL paper proposed an MLP policy that outputs a single output noise action, which is copied 10 times (π_0 -DROID has a default action length of 10). Unlike the original DSRL paper, we parameterize the policy with a transformer backbone so that it can output a unique noise embedding for each action in the 8-length action chunk—we found this action chunked version to perform more meaningful exploration and thus learn quicker. We copy the first 2 noise actions of the action chunk to fill the remaining 2 noise timesteps for π_0 -DROID, but we only execute 8 actions out of 10.

Similarly, the Q function takes as input an 8-length action chunk but otherwise follows a standard Q-function formulation. The network architectures follow the same one used in the Offline RL experiments, whose details are in Table F.23. SAC algorithm hyperparameters are detailed below.

Table F.18: Automatic Online RL. DSRL SAC hyperparameters.

Hyperparameter	Value
Batch size	64
Target update rate τ	0.005
Discount factor γ	0.995
Target update interval	1
Number of critics	4
Critics sampled per update	2
Pooled critic features	True
Actor updates per train step	10 (single), 12 (multi)
Critic updates per actor update	4 (single), 2 (multi)
Actor learning rate	5×10^{-5}
Critic learning rate	1×10^{-4}
Entropy coeff learning rate	1×10^{-4}
Target entropy	0
Learning starts	1200 steps
Training steps	10000 steps

We run RL for 10000 environment steps, corresponding to about 40 minutes of total real-

world experiment time. We train after each episode rather than after each step, and we design an *asynchronous* reward relabeling pipeline that relabels rewards in the replay buffer using reward models after they’ve been added, to prevent reward-related latency.

Success Detection. Success detection and episode termination come from the reward models. With Robometer, we use the model’s success prediction probability and threshold it so that if the last timestep’s success detection probability is > 0.6 , the episode is marked as a success and terminated. With RoboReward, we use its discrete reward predictions where we mark the episode as successful if it predicts 5/5 reward. If no success is detected by 240 timesteps, the episode terminates and resets.

For the multi-stage experiment, we run for 200 timesteps per stage. If the first stage (corn in pot) fails after 200 timesteps, we reset. If the reward model detects success (as in single-stage) in the first stage, we advance to the second stage, where the episode timeout is reset back to 200 timesteps. If there’s a failure in the second stage, we still reset the entire scene back to before the first stage was completed. Each stage advance simply updates the language instruction for π_0 and the task index for the DSRL policy/Q function.

Environment Reward. For the single-stage setup, the base reward for the task is $-1/0$, where -1 is given at every step except success, where 0 is given. This base reward is added to the reward predictions $\in [0, 1]$, thus bounding the per-step reward to be $[-1, 1]$, where the only possible positive reward signal comes at the end of a successful trajectory.

In the multi-stage setup, the base reward when the policy is in the first stage is -2 at each timestep, added to the reward predictions from the reward model. When the policy enters the second stage, the base reward is $-1/0$ just like in the single-stage setup. This simple reward design always encourages the policy to advance to the next stage, even under possibly suboptimal rewards.

Results. Our single-stage results in [Figure 8.6](#) show a 55% improvement in performance of Robometer over RoboReward after 10k online RL steps. A key failure mode of RoboReward is its

tendency to predict high rewards even when the robot is executing the wrong task. In cluttered environments, RoboReward frequently assigns a maximum reward (5/5) when the robot manipulates objects that are not the target bowl. This leads to a large number of false positives, as quantified in Table F.19.

Table F.19: Automatic Online RL reward relabeling. True positives (TP) and false positives (FP) for reward predictions during online RL.

Method	True Positives	False Positives
RoboReward	6	45
Robometer	18	0

Finally, in our multi-stage results in Figure 8.6, RoboReward does not improve the π_0 policy at all. This failure to improve stems from how the π_0 policy tends to fail the first subtask, putting the corn in the pot, by dropping it *near* but not in the pot. RoboReward often falsely assigns a 5/5 score to these failed states, allowing the policy to move onto the next stage. As a result, during evaluation, the RoboReward-trained DSRL policy often drops the corn into various parts of the dish rack rather than into the pot. Robometer, on the other hand, rarely assigns these states as successful, enabling the policy to learn more.

F.5.3 Failure Detection

For failure detection, we evaluate our model on MIT Franka dataset. For quantitative analysis in Table F.20 and Figure F.4, we mark a trajectory as success if success probability is higher than or equal to 0.5, and failure based on our progress monitoring strategy explained in this section.

Baselines. In addition to RoboReward-4B, VLAC, and GPT-5-mini, we compare against an uncertainty-based baseline, following [101, 125]. *Token-Uncertainty* baseline estimates predictive uncertainty by computing entropy over generated language tokens. We compute token uncertainty on evaluation trajectories using the π_0 -FAST DROID policy [239].

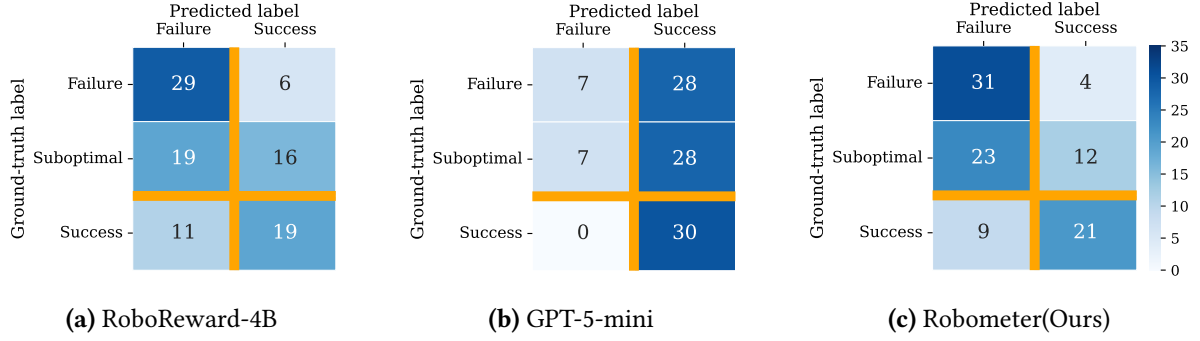


Figure F.4: Failure Detection OOD confusion matrices with ternary ground truth and binary prediction. Rows indicate ground-truth execution outcomes (*failure*, *suboptimal*, *success*), while columns indicate binary predictions (*predicted failure* vs. *predicted success*). *Suboptimal* trajectories correspond to executions that make partial progress but do not complete the task. Suboptimal trajectories are treated as failures in our quantitative evaluation, as emphasized with the horizontal orange divider in the above confusion matrices. Color intensity reflects the number of trajectories.

Discussion. Figure F.4 visualizes out-of-distribution failure detection using an asymmetric 3×2 confusion matrix: ground-truth labels include *failure*, *suboptimal*, and *success*, while predictions are binary (*failure* vs. *success*). This view separates two practically distinct error types: (i) missed failures/suboptimal executions (mass in the right column for the top two rows), and (ii) false alarms on successful executions (mass in the left column for the bottom row). Across methods, Robometer allocates more mass to correctly flagging both *failure* and *suboptimal* trajectories as failures, while maintaining relatively few false alarms on *success* trajectories, consistent with its higher average F1 in Table 8.4. In contrast, GPT-5-mini concentrates mass in the *predicted success* column for *failure* and *suboptimal* rows, indicating a conservative bias that avoids false positives but misses many non-success outcomes, whereas RoboReward-4B is intermediate with improved failure sensitivity but still more confusion between *suboptimal* and *success* than Robometer.

Failure detection with progress monitoring. Table F.20 compares our model against baselines on three evaluation metrics: true positive rate (TPR; correctly detecting failures), true negative rate (TNR; correctly identifying successful executions), and F1 score. Robometer consistently outperforms baselines, resulting in higher average F1 score. Token-Uncertainty and GPT-5-mini baselines attain perfect TNR but extremely low TPR, indicating a strong bias toward predicting

Table F.20: Failure detection performance. We report true positive rate (TPR; correctly detecting failures), true negative rate (TNR; correctly identifying successful executions), and F1 score.

Task	Token-Unc.			VLAC			GPT-5-mini			RoboReward-4B			Robometer		
	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1	TPR	TNR	F1
move banana	0.00	1.00	0.53	0.89	0.35	0.45	0.31	1.00	0.48	1.00	0.62	0.91	1.00	0.75	0.94
move mouse	0.00	1.00	0.50	1.00	0.00	0.00	0.80	1.00	0.89	0.80	0.75	0.80	1.00	0.75	0.91
pour pebble	0.00	1.00	0.32	0.88	0.00	0.00	0.14	1.00	0.25	0.57	1.00	0.73	0.71	1.00	0.83
fold towel	0.00	1.00	0.58	0.95	0.10	0.16	0.15	1.00	0.27	0.31	0.67	0.40	0.54	0.56	0.58
pull tissue	0.00	1.00	0.43	1.00	0.00	0.00	0.00	1.00	0.00	0.55	0.00	0.57	0.73	0.50	0.76
put spoon	0.00	1.00	0.22	1.00	0.00	0.00	0.14	1.00	0.25	0.57	1.00	0.73	0.57	1.00	0.73
stir pot	0.00	1.00	0.47	0.94	0.00	0.00	0.09	1.00	0.17	0.91	1.00	0.95	0.82	1.00	0.90
Average	0.00	1.00	0.48	0.95	0.10	0.16	0.20	1.00	0.33	0.69	0.63	0.74	0.77	0.70	0.81

success and missing failures. VLAC exhibits the opposite behavior: it frequently flags trajectories as failures, achieving high true positive rates but low true negative rates due to many false positives on successful executions, which results in lower F1 scores. RoboReward-4B lies between these extremes, with improved failure sensitivity over GPT-5-mini but more confusion between *suboptimal* and *success* trajectories than Robometer.

We provide qualitative examples of failure detection using progress monitoring across different failure categories in Figures F.5–F.7. Specifically, we compute the Pearson correlation between progress values and time over a sliding window, and flag a failure at the first timestep where this correlation becomes lower than a threshold. This captures both irreversible failures, where rewards sharply decrease after an error (e.g., object drops), and insufficient-progress failures, where rewards stagnate or regress over time. If no such failure is detected or if our model predicts success, the trajectory is classified as successful. Our failure evaluation dataset includes both *irreversible failures* (e.g., object drops or spills) and *insufficient-progress failures*, where the robot stalls, oscillates, or terminates execution before completing the task. We additionally highlight *semantic failures*, where the robot executes a physically plausible behavior that violates the task instruction. All qualitative examples use window size of 5 besides Figure F.5 (b) which uses window size of 9. We use correlation threshold of -0.5 in all examples.

Across all cases, our reward model identifies failures by detecting regressions or stagnation in

predicted task progress. For irreversible failures (see Figure F.5), the predicted progress increases initially but drops sharply after the terminal event (e.g., dropping or spilling), leading to early failure detection. In semantic failures (see Figure F.6), progress remains consistently low despite smooth execution, reflecting instruction-level mismatch. For insufficient-progress failures (see Figure F.7), the progress signal plateaus or oscillates without converging to success, allowing the model to flag failure even in the absence of abrupt terminal events.

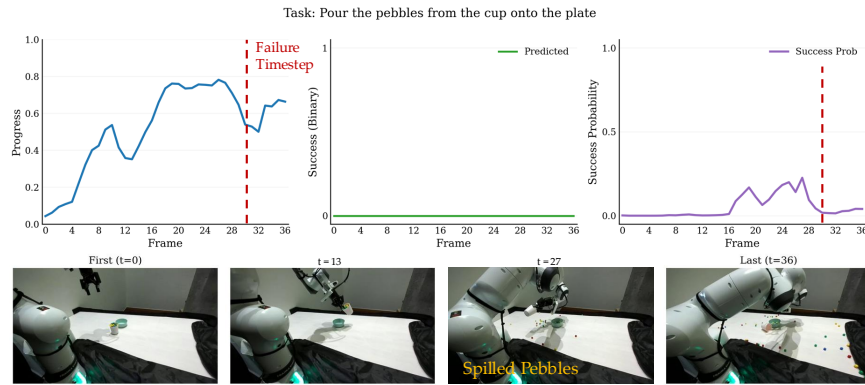
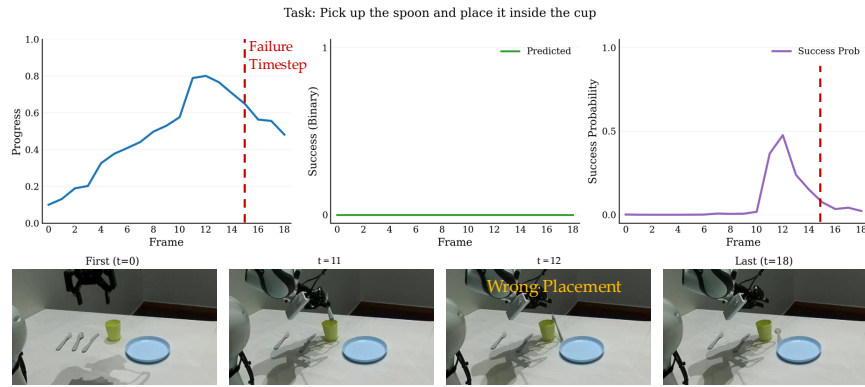
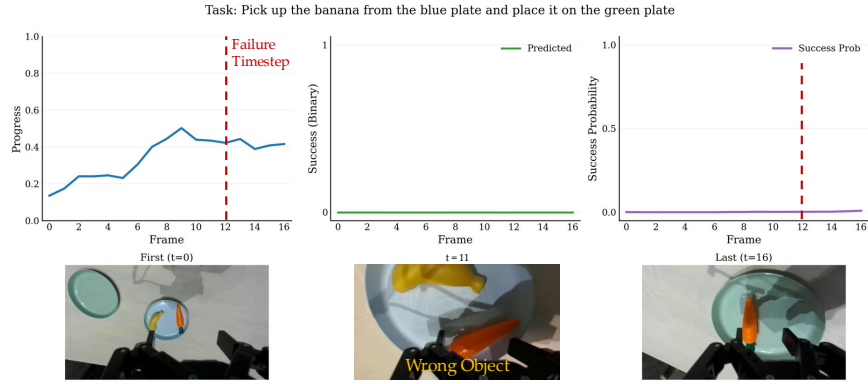
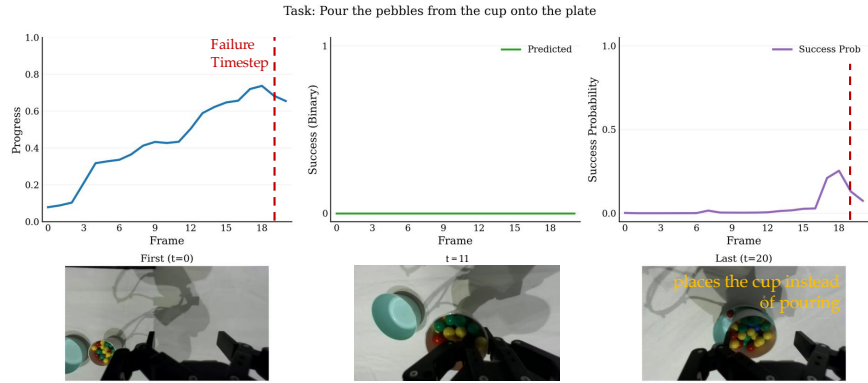


Figure F.5: Irreversible failures. Terminal events such as drops or spills cause a sharp regression in predicted task progress, which our model reliably flags as failures shortly after the event.



(a) Wrong object grasped despite correct motion pattern.



(b) Instruction-inconsistent action (placing the cup instead of pouring).

Figure F.6: Semantic failures. The robot executes smooth and physically plausible trajectories but violates the task instruction, resulting in persistently low predicted progress and failure detection without an abrupt terminal event.

Figure F.7: Insufficient-progress failures. Although the robot continues to move, predicted progress stagnates or oscillates without converging to success, enabling detection of failures caused by stalling or premature termination. Execution stalls with plateaued progress in this example.

F.5.4 Data Filtering & Retrieval

Procedure and discussion. We have 50 trajectories in our play dataset, where each trajectory consists of five tasks executed in random order: *uncap the red pen*, *open the bottle*, *open the red drawer*, *stir the pot*, and *unzip the pencil case*, collected using the Trossen Stationary AI bimanual setup. The dataset includes three camera views: one top-down view and one wrist-mounted camera for each end effector. Example images from all camera views are shown in Figure F.8. Following prior work [120, 215], we first segment each trajectory into subtrajectories based on end-effector velocities and changes in gripper states. In practice, we found that explicitly incorporating gripper state changes produces cleaner segmentation of different tasks. After segmentation, the play dataset yields a total of 615 subtrajectories. If subtrajectories were uniformly distributed across tasks, this would correspond to 123 relevant segments per task; to ensure conservative evaluation, we retrieve 100 subtrajectories per task.

We compare our method against three baselines: RoboReward, pre-trained SigLIP [330], a vision-language model trained with contrastive image-language supervision, and a retrieval-specific baseline, STRAP [215]. We chose to include STRAP as it is one of the few viable approaches for zero-shot data filtering and retrieval; alternative methods either require policy roll-outs [5], larger amounts of task-specific data [301], or training additional task-specific retrieval modules [75, 116, 194, 336].



Figure F.8: Data filtering & retrieval scene configuration from all cameras.

Given a task language instruction, we perform retrieval using different reward models as fol-

lows. For RoboReward and Robometer-Prog, we first compute per-timestep reward predictions for each subtrajectory conditioned on the instruction. We then calculate the value-order correlation of each subtrajectory using the predicted rewards and select the top 100 subtrajectories with the highest VOC scores. When using the preference-based variant of our method (Robometer-Pref), we construct pairwise comparisons between all subtrajectories, aggregate the results into a win matrix, and rank subtrajectories according to estimated pairwise preferences. The top 100-ranked subtrajectories are then selected. For SigLIP, we compute the average vision–language similarity between each subtrajectory and the instruction and select the top 100 subtrajectories with the highest average similarity scores. We use wrist-camera images for retrieval in all methods, as they yield the highest-quality retrievals for all methods.

For $\pi_{0.5}$ policy fine-tuning via LoRA [186, 202], we directly use the retrieved 100 subtrajectories together with the corresponding task instruction. During fine-tuning, we include observations from all camera views, as well as states and actions. Results reported in Figure 8.8(b) use a strict success metric: a trial is counted as successful only if the robot fully completes the task. This criterion partially explains the low success rates observed for the baselines. Qualitative inspection of learned behaviors reveals clear differences between retrieval methods. Policies trained on SigLIP-retrieved data often learn to approach task-relevant objects (e.g., reaching the drawer) but fail to complete the task. In contrast, policies trained on RoboReward-retrieved data frequently exhibit random or unstable behaviors, consistent with a higher proportion of unrelated subtrajectories in the retrieved set.

Additional experiments. We additionally evaluate retrieval quality in a controlled setting where ground-truth trajectory labels are available. Instead of using the play dataset, we consider the Trossen subset of RBM-EVAL-OOD, which contains six tasks with known numbers of successful, suboptimal, and failed trajectories. We perform retrieval for two representative tasks—*open the red drawer* and *unzip the pencil case*—each of which contains three successful, two suboptimal, and two failure trajectories. Using the same retrieval procedures described above, we retrieve five

trajectories per task and assess their quality. Results are shown in Table F.21. Robometer consistently retrieves higher-quality trajectories, while baselines either fail to retrieve task-relevant trajectories or select failure cases. These results highlight Robometer’s ability to both distinguish between tasks and differentiate levels of execution quality.

Table F.21: Trajectory retrieval analysis. Top-5 retrieval quality summary per task. Counts indicate how many of the top-5 retrieved trajectories are labeled as success (Succ), suboptimal (Subopt), failure (Fail), or unrelated (Unrel).

Task	Method	#Succ	#Subopt	#Fail	#Unrel
Open the red drawer	SigLIP	2	2	1	0
	RoboReward	2	1	2	0
	STRAP	2	2	1	0
	Robometer-Prog	3	2	0	0
	Robometer-Pref	3	2	0	0
Unzip the pencil case	SigLIP	2	1	2	0
	RoboReward	0	0	2	3
	STRAP	2	2	1	0
	Robometer-Prog	3	2	0	0
	Robometer-Pref	2	2	1	0

F.5.5 Combining Noisy and Expert Trajectories via Offline RL

Procedure and data collection. We study how different reward formulations affect downstream policy learning when training on a mixture of expert and noisy trajectories using offline reinforcement learning. Specifically, we compare three reward settings: (i) a sparse terminal reward, (ii) rewards predicted by RoboReward, and (iii) rewards predicted by Robometer.

We evaluate offline RL on the SO-101 robot platform under two settings across two manipulation tasks: *Put the bread in the oven* and *Put the red bowl on the blue plate*. **Setting 1** is a clean single-task setting that evaluates *Put the bread in the oven*; **setting 2** is a cluttered multi-task setting that evaluates *Put the red bowl on the blue plate*. We visualize these settings in Figure 8.7.

For each task, we collect a mixed-expertise offline dataset containing both expert demon-

strations and failed executions. In **Setting 1**, we collect 30 successful trajectories and 45 failed trajectories for *Put the bread in the oven*. In **Setting 2**, we collect 20 successful trajectories and 15 failed trajectories each for 3 separate tasks: *Put the marker in the pen cup*, *Put the red cup on the purple coaster*, and the evaluation task *Put the red bowl on the blue plate*. With offline RL, transitions from other tasks, whether the data is suboptimal or expert, should be useful for the *Put the red bowl on the blue plate* task as they share a similar cluttered scene.

Observations include images from both an external camera and a wrist-mounted camera, as well as proprioceptive states. During offline RL training, we use images from both camera views, while reward estimation is performed using the external camera view only.

Offline RL setup. We follow an offline RL training setup similar to ReWiND [333]. For all methods, we train policies using Implicit Q-Learning (IQL) [161] with identical actor, critic, and value network architectures, optimization hyperparameters, and datasets.

IQL objectives. IQL learns a value function $V_\psi(s)$, an ensemble of action-value functions $\{Q_{\theta_i}(s, a)\}_{i=1}^N$, and a policy $\pi_\phi(a | s)$ from an offline dataset $\mathcal{D} = \{(s, a, r, s')\}$. Each critic is trained by minimizing the temporal-difference loss $\mathcal{L}_Q(\theta_i) = \mathbb{E}_{\mathcal{D}}[(Q_{\theta_i}(s, a) - (r + \gamma V_\psi(s')))^2]$. The value network is trained via expectile regression toward the critic ensemble using $\mathcal{L}_V(\psi) = \mathbb{E}_{\mathcal{D}}[\rho_\tau(\min_i Q_{\theta_i}(s, a) - V_\psi(s))]$, where $\rho_\tau(u) = |\tau - \mathbb{I}(u < 0)|u^2$. The policy is learned via advantage-weighted regression with objective $\mathcal{L}_\pi(\phi) = \mathbb{E}_{\mathcal{D}}[\exp(A(s, a)/\beta) \log \pi_\phi(a | s)]$, where the advantage is defined as $A(s, a) = \min_i Q_{\theta_i}(s, a) - V_\psi(s)$ and β is the advantage temperature.

To account for differences in reward sparsity and temporal structure, we sweep the discount factor $\gamma \in \{0.9, 0.95, 0.99\}$ for each reward setting. All other IQL hyperparameters and actor-critic architecture details are held fixed across methods and summarized in Tables F.22 and F.23. For each method, we select the best-performing checkpoint based on validation performance.

Reward specification. RoboReward provides a sparse, trajectory-level progress signal. To enable its use in IQL, we convert RoboReward outputs into a per-frame dense reward by querying

the model on partial trajectory prefixes $o_{1:t}$ and assigning the resulting prediction as the reward at timestep t . In contrast, our reward model directly produces dense, temporally aligned rewards over video sequences, which we use without additional post-processing. Aside from this difference in reward construction, all other aspects of offline RL training are kept identical across methods.

Table F.22: SO-101 offline RL experiment. Hyperparameters used for Implicit Q-Learning (IQL) in offline policy training.

Hyperparameter	Value
Batch size	256
Target update rate τ	0.005
Discount factor γ	$\{0.9, 0.95, 0.99\}$
Target update interval	1
Advantage temperature	2
Expectile	0.7
Advantage clipping	100.0
Policy extraction method	AWR [238]
Number of critics	5
Critics sampled per update	2
Pooled critic features	True
Updates per train step	1
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Value network learning rate	3×10^{-4}
Weight decay	0.0

F.6 Low-rank Fine-tuning Robometer-4B

RoboFAC dataset. We fine-tune and evaluate on RoboFAC, a video-based dataset designed for robotic failure analysis and correction. RoboFAC spans 16 tasks across 53 scenes and provides 78K video QA pairs annotated over 10,722 trajectories. The dataset contains both failures and successes: 9,440 failure trajectories and 1,282 successful trajectories. Failures are collected in both simulation and the real world, with 8,960 simulated failures and 480 real-world failures;

Table F.23: SO-101 offline RL model architectures. Transformer-based actor and critic architectures used for offline IQL training. Both networks share a lightweight Transformer encoder, followed by task-specific MLP heads.

Component	Actor	Critic
Transformer Encoder		
Model dimension (d_{model})	256	256
Number of heads	8	8
Encoder layers	6	6
Transformer dropout	0.0	0.0
Transformer activation	GELU	GELU
Layer normalization	False	True
Pooling strategy	–	First token
Feature Processing MLP		
Hidden dimensions	[512, 512]	[768, 512]
Activation	ReLU	ReLU
Dropout	0.0	0.0
Output Head		
Output hidden dims	None	None
Action squashing	Tanh	–
Deterministic policy	False	–
Log-std init	0	–
Log-std range	[−20, 2]	–

additionally, the dataset includes 1,160 simulated successes and 122 real-world successes.

LoRA fine-tuning Robometer-4B. We keep our pre-trained Robometer-4B reward model backbone fixed and train LLM LoRA adapters (76M parameters) while also fine-tuning the MLP prediction heads (progress, preference, and success). We train for 500 steps with a batch size of 8 on a single NVIDIA RTX A6000 for approximately 8 hours of wall clock time.

Baselines. To isolate the benefit of initializing from our pre-trained Robometer-4B checkpoint versus starting from scratch, we include two Qwen3-VL baselines fine-tuned on the same dataset. First, we perform LoRA fine-tuning starting from Qwen/Qwen3-VL-4B-Instruct and train the prediction heads from random initialization. We also fully fine-tune all parameters of Qwen/Qwen3-VL-4B-Instruct, again training the prediction heads from random initialization.

Results. Table 8.2 reports offline reward evaluation on RoboFAC. In the zero-shot setting, Robometer-4B already achieves strong correlation with ground-truth progress (VOC $r=0.652$, Kendall $\tau=0.436$), indicating meaningful transfer without any RoboFAC supervision. After adaptation, Robometer-4B provides a substantially better initialization than training the base VLM from scratch. Full fine-tuning from our checkpoint reaches VOC $r=0.884$ and Kendall $\tau=0.802$, outperforming the strongest from-scratch baseline by +21.6% in VOC and nearly an order of magnitude in Kendall τ . Notably, LoRA on Robometer-4B attains essentially the same performance, indicating that the gains primarily come from our pre-trained initialization rather than full-parameter adaptation.

F.7 Robometer with Model-Based RL

Setup. For this proof-of-concept experiment, we integrate Robometer into a model-based algorithm by using it to rank candidate trajectories sampled from the world model.

We adopt the DreamZero [320] world model checkpoint trained on DROID to use in an identical DROID setup to the online RL experiments, albeit with a different task.

Given 3 camera observations, proprioceptive states, and a language instruction, DreamZero simultaneously generates candidate future observation sequences (multiple video frames span-

ning 1.6 seconds) with associated action chunks of length 24.

We compare DreamZero with and without Robometer ranking candidate trajectories. DreamZero without Robometer directly generates a single candidate future observation sequence from which we extract the action chunk. With Robometer, we generate 6 candidate observation sequences at each inference step, rank them with the progress output from Robometer, and then execute the action sequence extracted from the highest-ranked generated observation sequence.

In total, generating a single action chunk of length 24, including the world model forward pass for the 6 candidate observation sequences, takes approximately 28 seconds on 1 H200, resulting in a single trajectory taking around 3 minutes to execute including robot execution time. The vast majority of this inference time is from the world model; Robometer’s forward pass only takes around 0.6-1 seconds.

Results. We evaluate on a “put the ice cream in the pink plate” task in an extremely cluttered scene with many possible receptacle plates of different colors. DreamZero typically places the ice cream cone in the wrong plate, while integrating Robometer corrects for this mistake. Overall, results in Figure F.9 demonstrate a 3.5× improvement in success rate evaluated over 10 trials with Robometer.

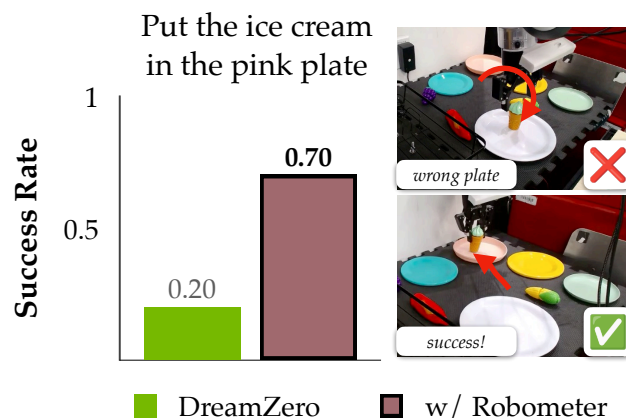


Figure F.9: Model-Based RL with Robometer integrated into DreamZero [320]. In this cluttered scene, Robometer improves DreamZero’s performance from 20% success rate to 70%.