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

Anthony Liang[1,*], **Pavel Czempin**[1,*], **Matthew Hong**[1], **Yutai Zhou**[1],
Erdem Bıyık[1,2†], **Stephen Tu**[1,2†]
[1]Department of Computer Science, University of Southern California
[2]Department of Electrical and Computer Engineering, University of Southern California
anthony.liang@usc.edu

Abstract: Learning robot policies using imitation learning requires collecting large amounts of costly action-labeled expert demonstrations, which fundamentally limits the scale of training data. A promising approach to address this bottleneck is to harness the abundance of *unlabeled* observations—e.g., from video demonstrations—to learn latent action labels in an unsupervised way. However, we find that existing methods struggle when applied to complex robot tasks requiring fine-grained motions. We design *continuous latent action models* (CLAM) which incorporate two key ingredients we find necessary for learning to solve complex continuous control tasks from unlabeled observation data: (a) using *continuous* latent action labels instead of discrete representations, and (b) *jointly training* an action decoder to ensure that the latent action space can be easily grounded to real actions with relatively few labeled examples. Importantly, the labeled examples can be collected from non-optimal play data, enabling CLAM to learn performant policies *without* access to any action-labeled expert data. We demonstrate on continuous control benchmarks in DMControl (locomotion) and MetaWorld (manipulation), as well as on a real WidowX robot arm that CLAM significantly outperforms prior state-of-the-art methods, remarkably with a $2-3\times$ improvement in task success rate compared to the best baseline. Videos and code can be found at clamrobot.github.io. [1]

Keywords: Latent Action Models, Self-supervised Pretraining

1 Introduction

Large pretrained foundation models in natural language processing [1, 2] and computer vision [3, 4] have demonstrated impressive capabilities in a variety of challenging downstream tasks. A key ingredient to the success of foundation models has been the abundance of Internet-scale pretraining text and image corpora, which enable foundation models to learn rich representations [5]. Unfortunately, replicating this success in the context of training capable robotics policies remains an open challenge. While many efforts have been made to create larger robotics datasets [6, 7, 8], these efforts still rely on manual data collection and hence are unlikely to scale to datasets of the same order of magnitude as in computer vision and NLP domains.

Videos from Internet sources such as YouTube contain information that can potentially inform robots about the physical world and how to perform various tasks. Leveraging these data sources would allow for more cost-effective scaling of robot policies than training workers to teleoperate robot hardware. While in-the-wild videos are abundant and freely available, it is non-trivial how to effectively utilize them for learning robot policies. In this work, we focus on ameliorating one of the primary issues with learning from video data: the lack of low-level action labels for supervising modern robot learning paradigms such as reinforcement learning and imitation learning (IL).

[1] $*$ Equal contribution, † Equal advising

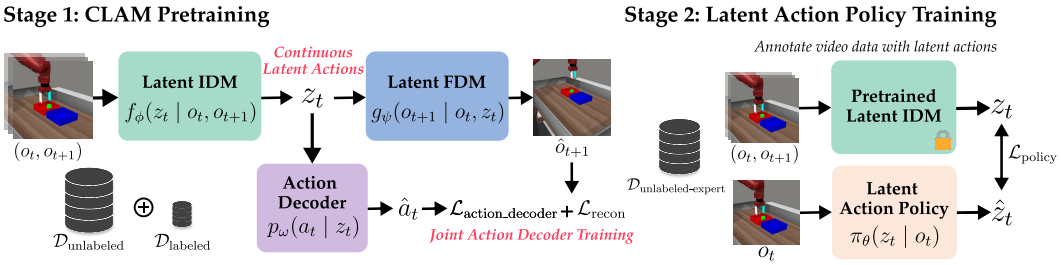

Figure 1: **Overview of CLAM.** CLAM consists of a **latent inverse dynamics model**, f_ϕ, which infers the latent action between a transition and **latent forward dynamics model**, g_ψ, which predicts the future observation conditioned on the latent action. CLAM learns a latent action space through 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.

Prior works on learning from observation-only data either require off-the-shelf models to annotate visual features, limiting the expressivity of the policy representation [9, 10], or require a large amount of action-labeled expert data for downstream imitation learning [11, 12, 13]. Instead, we adopt a self-supervised framework for training an inverse dynamics models (IDMs) to annotate observation-only data with action labels which can subsequently be used for IL. Since these action labels reside in some latent space, we propose to jointly train an action decoder that grounds these *latent actions* to executable environment actions. We show that this action decoder can be trained on a *small* amount of labeled play data, eliminating the need for expensive expert data collection [14].

In this work, we identify several key architectural improvements that greatly improve the paradigm of learning from observation-only data in continuous control settings. Our contributions are:

1. We demonstrate that latent actions should not be discretized, contrary to prior works, but rather remain ***continuous*** to be effective for fine-grained robot control tasks.
2. We propose to ***jointly train*** a latent action model and action decoder for grounding latent actions to the environment. We demonstrate that a jointly trained continuous latent action space improves downstream policy performance by $2-3\times$ on continuous control tasks in DMControl, MetaWorld, and on a real WidowX robot arm.
3. We learn a performant control policy *without* ever training on *labeled expert* demonstrations, instead leveraging a small amount of action labels from *random* or *play* data.

2 Problem Formulation

Learning from in-the-wild video data is challenging and an active area of research in robotics. In the scope of this paper, we focus on *single-task, single-embodiment* settings with observation-only data and leave the challenge of learning from real in-the-wild videos for future work.

In this setting, a large offline dataset of observation-only data of varying expertise levels, $\mathcal{D}_{\text{unlabeled}}$, is provided. This dataset is representative of in-the-wild video data which is not always optimal and lacks action labels. We also assume access to a *small* dataset of labeled data, $\mathcal{D}_{\text{labeled}}$ for grounding to true robot actions. This is the only labeled data available. Crucially, $\mathcal{D}_{\text{labeled}}$ does not necessarily need to be expert, but rather can be *random* or *play* data. This enables CLAM to be more scalable than the existing methods in the literature, as non-expert play data is significantly easier to collect than expert task demonstrations [14]. Finally, we assume a subset of the *unlabeled* data comes from an expert: $\mathcal{D}_{\text{unlabeled-expert}} \subseteq \mathcal{D}_{\text{unlabeled}}$. This assumption is necessary for imitation learning, as the remaining data do not carry any information to learn the task. *We note that our methods and all baselines have access to these same datasets*. However, methods differ in whether they require labeled data in various parts of their pipeline.

3 Continuous Latent Action Models

We introduce *continuous latent action models* (CLAM), a scalable approach for training continuous control policies from unlabeled observation data. We show an overview of the CLAM architecture in Figure 1 and summarize the training procedure in Algorithm 1 (Appendix C). CLAM consists of two stages. In **Stage 1** (Section 3.1), we train a latent action model (LAM) for relabeling observation-only data. We then use this LAM in **Stage 2** (Section 3.2) to train a latent action policy.

3.1 Latent Action Model Training

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

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 1, the training signal comes from reconstruction of the future observation, i.e. $\mathcal{L}_{\text{recon}} = \text{MSE}(\hat{o}_{t+1}, o_{t+1})$ where $\hat{o}_{t+1}$ is the prediction from the FDM. Our encoder/decoder architecture induces an information bottleneck that ensures learning a meaningful, compact action representation rather than shortcut solutions. While prior works [15, 16] discretize latent actions from the IDM using Vector Quantization [VQ; 17], 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 action space.

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. Prior work [15] 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. For continuous action spaces, we will show that the latent action space learned by forward reconstruction alone is difficult to decode to real-world actions.

To address this, we propose ***joint training*** of the action decoder and the latent action model to regularize the learned latent action space, ensuring that it 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 the $\mathcal{D}_{\text{labeled}}$ *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.

3.2 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., $\mathcal{D}_{\text{relabeled-expert}} = \{(o_1^i, z_1^i, o_2^i, z_2^i, \dots, z_{T-1}^i, o_T^i)\ \forall \tau^i \in \mathcal{D}_{\text{unlabeled-expert}}\}$.

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 $\mathcal{D}_{\text{relabeled-expert}}$. During inference time, our

learned policy predicts the latent actions given an observation, which the action decoder will decode into an environment action. Pseudocode of the inference loop is provided in Appendix C.

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 pretrained 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 [18], 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 4 that similar positive transfer also occurs for CLAM when learning latent action policies.

Avoiding trivial solutions for latent action labels. Our self-supervised reconstruction objective for training LAM could potentially be vulnerable to locally-optimal shortcuts such as learning an identity mapping. We provide an extensive discussion on how we mitigate these kinds of issues along with analysis demonstrating that CLAM does not degenerate in Appendix B.

4 Experimental Setup

We compare CLAM to several state-of-the-art baselines using both state- and image-based observations. We show quantitative results in simulated locomotion tasks in DMControl [19], robot manipulation tasks in MetaWorld [20], and on a real WidowX robot arm. For DMControl and MetaWorld, $\mathcal{D}_{\text{unlabeled-expert}}$ comes from trained RL agents. In our real robot experiments, we use expert trajectories with the action labels removed for proof-of-concept. We also conduct additional experiments in the CALVIN [21] benchmark in Appendix A.3. For more task details, we refer to Appendix E.

- **DMControl.** The D4RL [22] benchmark comprises of replay buffers from trained RL agents of varying expertise. For training CLAM, we use 1000 trajectories from the `medium-replay` dataset as $\mathcal{D}_{\text{unlabeled}}$ and subsample trajectories for $\mathcal{D}_{\text{labeled}}$. This data split contains trajectories from the online RL replay buffer up until the policy reaches a *medium* level of performance.
- **MetaWorld.** We use TD-MPC2 [23] to train single-task RL agents and collect the replay buffer data. We use the full dataset (1000 trajectories of mixed expertise) for training our LAM and a separate `random-medium` dataset similar to DMControl for $\mathcal{D}_{\text{labeled}}$.
- **WidowX Robot Arm.** To demonstrate the scalability of CLAM to more realistic scenarios, we evaluate on four real world manipulation tasks using a WidowX robot arm shown in Figure 2. We collect a task-agnostic dataset of $\sim$50k transitions for $\mathcal{D}_{\text{unlabeled}}$. We also have $\sim$30 expert demonstrations per task without action labels for $\mathcal{D}_{\text{unlabeled-expert}}$.

Baselines. To allow for a fair comparison, we reuse the same architectural components in our method and all baselines, with identical network architectures. These components are an IDM, an FDM, an action decoder/action head, and an MLP-based BC policy. We refer the reader to Appendix G and Appendix I for implementation details of CLAM and baseline methods.

- `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` **[11]**: 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` **[15]**: Latent action model with discrete, vector-quantized latent actions.
- `LAPA` **[13]**: 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*. Because the model is fine-tuned on this data, in the original paper, they use *labeled expert* data, which we do not have access to.
- `DynaMo` **[18]**: 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.

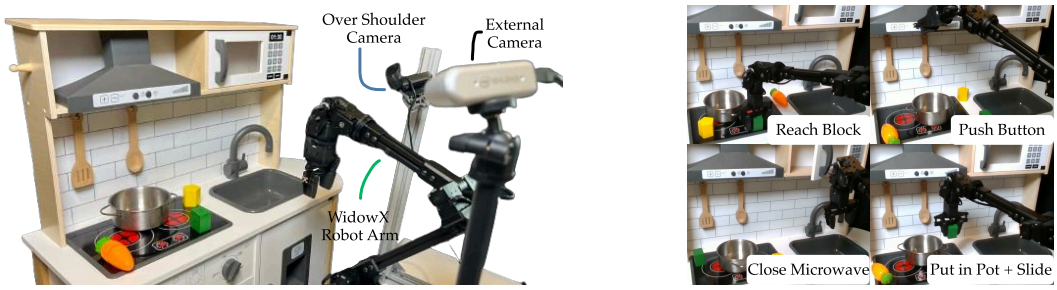

Figure 2: **WidowX Robot Arm Setup and Evaluation Tasks.** We evaluate the scalability of CLAM using four manipulation tasks (Right) on a WidowX robot arm (Left) in a toy kitchen setup [24].

- `MLP`-, `Transformer`-, `ST-ViViT`-**CLAM (Ours)**: Continuous latent action model with different parameterizations of the latent IDM and FDM with joint action decoder training.
- `BC-Expert` (`BC-E`): Privileged BC on $\mathcal{D}_{\text{unlabeled-expert}}$ with *ground-truth* action labels available, which are not available to other methods.

5 Results

The aim in our experiments is to study the efficacy of CLAM as a general approach to learn from action-less 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:

(Q1) How effective is CLAM at learning policies *without* action-labeled expert demonstrations?
(Q2) How important are *continuous* latent actions and *jointly* training the action decoder?
(Q3) Can CLAM scale to learn capable robot policies in 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 1 summarizes our results for state-based inputs on DMControl and Figure 3 for image-based results in MetaWorld. More results for state-based experiments are provided in Appendix A.1. CLAM improves upon the best baseline `VPT` by more than $2\times$ average normalized return on the DMControl (***locomotion***) tasks and around $2-3\times$ success rate on the MetaWorld (***manipulation***) tasks.

`BC-AL` using *action-labeled* data unsurprisingly does not perform well due to imitating suboptimal demonstrations. In several tasks, `Transformer-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 3.2). 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.

	HalfCheetah	Hopper
`BC-AL`	0.22 $\pm$0.05	0.35 $\pm$0.04
`LAPO`	0.12 $\pm$0.05	0.24 $\pm$0.03
`LAPA`	0.22 $\pm$0.05	0.30 $\pm$0.06
`DynaMo`	0.18 $\pm$0.03	0.22 $\pm$0.02
`VPT`	0.32 $\pm$0.04	0.41 $\pm$0.03
`MLP-CLAM`*	0.64 $\pm$0.05	0.64 $\pm$0.03
`TF-CLAM`*	**0.72 $\pm$0.04**	**0.81 $\pm$0.05**
`BC-Expert`	0.68 $\pm$0.02	0.76 $\pm$0.04

Table 1: **State-Based Results** on DMControl tasks. **Maroon** indicates best performance except `BC-Expert`, which uniquely has access to *labeled* expert data. Our methods are denoted with *.

All variants of CLAM outperform the best baseline `VPT` [11], 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 outperform state-of-the-art methods in our problem setting where only *play* data is available as action-labeled data, and expert data is action-less. 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 [21] avoiding the need for expensive task-specific data collection.

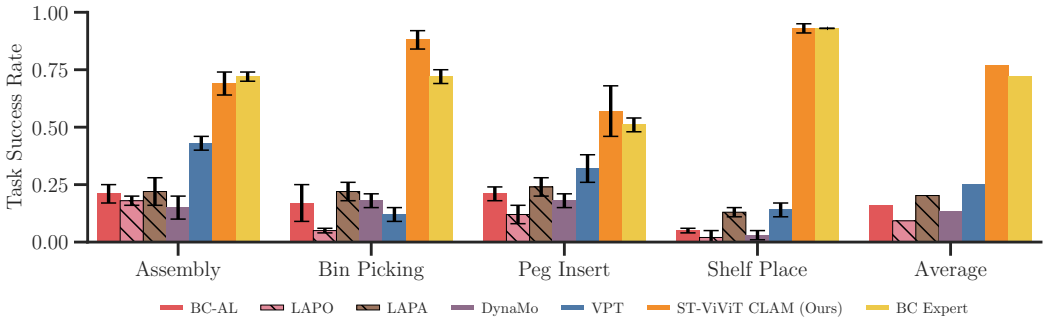

Figure 3: **MetaWorld Image-Based Experiments.** Task success rate over 50 evaluation rollouts across 3 random seeds using image-based inputs. All methods are trained using the same amount of action-labeled data (100 trajectories) for fair comparison. Since $\mathcal{D}_{\text{labeled}}$ is not necessarily expert data, all the baselines struggle to learn a performant downstream policy, while `MLP-CLAM` and `Transformer-CLAM` perform significantly better, with a $3\times$ improvement in task success over the best baseline. We denote baselines that use a *discrete* latent action space with hashed markers. We report results for `BC-Expert` which trains on the same amount of *expert* trajectories to represent the ideal performance BC achieves with ground truth action labels.

FINDING 2: Continuous latent actions and joint action decoder greatly improve performance for continuous control problems. Unlike `VPT`, the other baselines (`LAPO`, `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 [VQ; 17] to discretize the latent actions, including `LAPO` 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 `LAPO` and `LAPA`, which achieve 9% and 20%, respectively (cf. Figure 3). 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.

	Assembly	Bin Pick	Peg Insert	Shelf Place
Disc., ¬JT	0.15 ± 0.03	0.12 ± 0.02	0.18 ± 0.04	0.19 ± 0.03
Disc., JT	0.14 ± 0.04	0.14 ± 0.03	0.17 ± 0.03	0.16 ± 0.04
Cont., ¬JT	0.28 ± 0.04	0.18 ± 0.03	0.21 ± 0.08	0.26 ± 0.06
Cont., JT	$\mathbf{0.69} \pm 0.05$	$\mathbf{0.82} \pm 0.04$	$\mathbf{0.57} \pm 0.11$	$\mathbf{0.88} \pm 0.02$

Table 2: **CLAM Ablation Study.** Both continuous latent actions and joint training are necessary to improve task success. JT refers to *joint action decoder training*.

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 [16]. However, learning this mapping is more challenging for continuous actions. In Table 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` [15], 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).

FINDING 3: CLAM successfully learns without ever accessing *action-labeled expert* data in both simulation and real robot experiments. We perform our main-line experiments (Figure 3) 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 [14]. `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

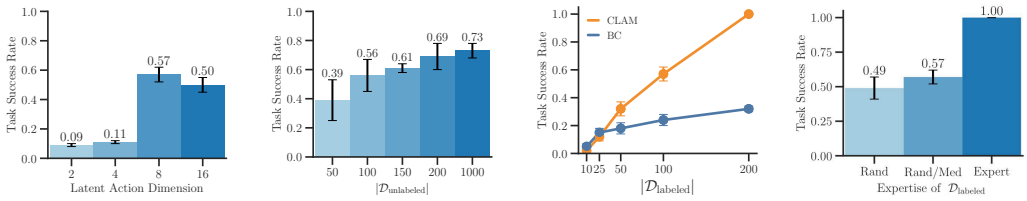

Figure 4: **CLAM Design Choices (a)** There are diminishing returns as we increase the latent action dimension. However, setting it too small limits the model's expressivity. **(b)** Policy performance continues to scale with $|\mathcal{D}_{\text{unlabeled-expert}}|$. **(c)** Latent action policy performance improves with more *non-expert* $\mathcal{D}_{\text{labeled}}$ while BC plateaus. **(d)** CLAM is robust to expertise level of $\mathcal{D}_{\text{labeled}}$.

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% 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 4d 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.

Real WidowX Robot Arm Experiments. We evaluate the scalability of CLAM to more realistic applications on a physical WidowX robot arm shown in Figure 2 (Left). Even though $\mathcal{D}_{\text{labeled}}$ 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.

	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**

Table 3: **Real Robot Results**. ST-CLAM significantly outperforms baseline methods across all tasks.

5.1 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. Appendix A.2 contains more ablation results.

Latent action dimension directly affects the model's expressivity. In Figure 4a, 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 $|\mathcal{D}_{\text{unlabeled-expert}}|$. In Figure 4b we demonstrate that CLAM's performance on the `Assembly` task improves as we increase $|\mathcal{D}_{\text{unlabeled}}|$. 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 $|\mathcal{D}_{\text{labeled}}|$ improves the action decoder accuracy and downstream performance. In Figure 4c we analyze the effect of varying $|\mathcal{D}_{\text{labeled}}|$ for training the action decoder. Unsurprisingly, as we increase $|\mathcal{D}_{\text{labeled}}|$, 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.

6 Related Work

Imitation from Observation. Learning robot policies from sequences of observations is often called *imitation from observation* [IfO; 25]. IfO poses the problem of learning from sequences that do not necessarily provide egocentric imitation learning data. Consequently, the observed data does not contain action labels and might differ in ways such as embodiment, environment, objects, and viewpoint. The goal is to mimic ways in which humans can learn, for example, by watching someone else demonstrate a task on video.

In this paper, we focus on the lack of action-labels in these observation sequences. Some approaches to IfO learn a policy directly from the observations, for example, by learning to translate and align observation sequences [25, 26, 27]. Alternatively, it is possible to estimate future observations and use learned or off-the-shelf methods for predicting robot control signals from observations [9, 10]. These methods often require the ability to perform online rollouts or use off-the-shelf computer vision tools. With CLAM, we aim to develop a general method without making any limiting. For example, methods that learn to directly predict keypoint movements [9], may fail when the action representation cannot easily be translated from these deltas. CLAM, on the contrary, is trained in an offline fashion and minimizes the assumptions about the ground-truth action space.

Supervised Learning of Inverse Dynamics Models. Assuming access to some amount of action-labeled data, it is possible to learn an inverse dynamics model in a supervised way. VPT [11] uses this IDM to label a larger unlabeled dataset. UniPi [12] finetunes a video prediction model and uses the IDM to predict robot actions from synthesized observations. One fundamental limitation of these approaches is that the IDM predicts actions in the *real* action space, and hence must be trained using action-labeled data. Consequently, these methods are bottlenecked by the number of labeled demonstrations or the budget required to collect them.

Latent Action Models. Prior works have shown that IDMs can be learned in an *unsupervised* way [28, 16, 15, 18, 13]. Both LAPA [13] and DynaMo [18] use unlabeled in-domain robotics demonstrations to train a latent action model that learns useful representations for downstream robot tasks. We find that these approaches have limited success in fine-grained manipulation due to the use of discrete latent actions reducing the expressivity of the learned latent space. Additionally, since these approaches learn an initial representation, but do not use the pre-training data to train a policy, every new task requires collecting action-labeled expert trajectories and running imitation learning.

Recent works LAPO [15] and Genie [16] combine a VPT-style approach with latent action models. Instead of learning the latent action model only for expressive representations, they directly use the IDM to label the unlabeled observations. Both works perform experiments only in video game-based environments with a small number of discrete actions. Our work finds the key architectural decisions to make this type of training feasible in fully continuous high-dimensional action spaces.

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

8 Limitations and Future Work

Data Regime. We emphasize that the findings in this paper consider a setting where practitioners want to reduce the cost of human labor by not needing *expert* teleoperated data. In regimes where this data is cheaply available for all tasks, other state-of-the-art methods could possibly achieve competitive or better results.

Data Efficiency. One limitation of CLAM is that it still requires diverse, labeled data—albeit non-expert—to capture the full action space for learning an accurate grounding model. In subsequent work, we plan to investigate solutions to improve the data-efficiency of CLAM. One potential direction is to experiment with different parameterizations of our latent action model, to further reduce the data needed to learn an accurate action decoder. Another possible solution is to impose structure on the learned latent action space by using a hybrid discrete and continuous latent action space. We could also use auxiliary training objectives such as multi-step forward prediction [29] or prior regularization like in variational autoencoders [30]. By enforcing a better latent structure, we can improve learning the mapping between latent and environment actions.We leave the investigation of these alternatives for future work.

In-the-Wild Internet Videos. To train large-scale foundation policies from in-the-wild Internet scale video data, a number of longstanding challenges still need to be addressed. Some of these include the embodiment gap between the human hand and robot end-effector, the visual domain gap in real-world scenes, and different camera angles, which are all beyond the scope of this work. As a result of these various challenges, we do not currently experiment with in-the-wild Internet videos of humans performing the same task. Instead, we demonstrate the efficacy of our approach by using real robot observation sequences without the action labels.

Embodiment Gap. We plan to address the embodiment challenge in future work. Most prior approaches to generalizing between different robot embodiments are limited to a fixed set of embodiments seen during training. Prior works such as CrossFormer [31] train fine-tuned action heads for different embodiments. 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 in a scalable way. To tackle the embodiment differences, one potential solution is to learn an embodiment model via contrastive learning and condition our latent action model on the learned embodiment embedding.

Latent Action Policy Architecture. In this work, we focus on learning a useful latent action space and use simple fully-connected layers to parameterize our latent action policy. In future work, we aim to explore more powerful policy architectures, such as ACT [32] and diffusion policies [33], to further maximize the performance of the downstream policy.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[4] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, X. Wang, X. Zhai, T. Kipf, and N. Houlsby. Simple open-vocabulary object detection. In *European Conference on Computer Vision*, 2022.

[5] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[6] H. Xiong, H. Fu, J. Zhang, C. Bao, Q. Zhang, Y. Huang, W. Xu, A. Garg, and C. Lu. Robotube: Learning household manipulation from human videos with simulated twin environments. In *Conference on Robot Learning*, 2023.

[7] Open X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[8] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[9] C. Wen, X. Lin, J. So, K. Chen, Q. Dou, Y. Gao, and P. Abbeel. Any-point trajectory modeling for policy learning, 2023.

[10] P.-C. Ko, J. Mao, Y. Du, S.-H. Sun, and J. B. Tenenbaum. Learning to act from actionless videos through dense correspondences. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Mhb5fpA1T0.

[11] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, 2022.

[12] Y. Du, S. Yang, B. Dai, H. Dai, O. Nachum, J. B. Tenenbaum, D. Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=bo8q5MRcwy.

[13] S. Ye, J. Jang, B. Jeon, S. Joo, J. Yang, B. Peng, A. Mandlekar, R. Tan, Y.-W. Chao, B. Y. Lin, L. Liden, K. Lee, J. Gao, L. Zettlemoyer, D. Fox, and M. Seo. Latent action pretraining from videos. In *International Conference on Learning Representations*, 2024.

[14] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, 2020.

[15] D. Schmidt and M. Jiang. Learning to act without actions. In *International Conference on Learning Representations*, 2023.

[16] J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, M. Lai, A. Mavalankar, R. Steigerwald, C. Apps, Y. Aytar, S. M. E. Bechtle, F. Behbahani, S. C. Chan, N. Heess, L. Gonzalez, S. Osindero, S. Ozair, S. Reed, J. Zhang, K. Zolna, J. Clune, N. de Freitas, S. Singh, and T. Rocktäschel. Genie: Generative interactive environments. In *International Conference on Machine Learning*, 2024.

[17] A. Van Den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 2017.

[18] Z. J. Cui, H. Pan, A. Iyer, S. Haldar, and L. Pinto. Dynamo: In-domain dynamics pretraining for visuo-motor control. In *International Conference on Machine Learning*, 2024.

[19] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

[20] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2020.

[21] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. In *IEEE Robotics and Automation Letters*, 2022.

[22] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[23] N. Hansen, X. Wang, and H. Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning*, 2022.

[24] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, 2023.

[25] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *International Conference on Robotics and Automation*, 2018.

[26] C. Yang, X. Ma, W. Huang, F. Sun, H. Liu, J. Huang, and C. Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. In *Advances in Neural Information Processing Systems*, 2019.

[27] F. Liu, Z. Ling, T. Mu, and H. Su. State alignment-based imitation learning. In *International Conference on Learning Representations*, 2020.

[28] W. Menapace, S. Lathuiliere, S. Tulyakov, A. Siarohin, and E. Ricci. Playable video generation. In *Conference on Computer Vision and Pattern Recognition*, 2021.

[29] A. Levine, P. Stone, and A. Zhang. Multistep inverse is not all you need. *arXiv preprint arXiv:2403.11940*, 2024.

[30] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 2019.

[31] R. Doshi, H. R. Walke, O. Mees, S. Dasari, and S. Levine. Scaling cross-embodied learning: One policy for manipulation, navigation, locomotion and aviation. In *Conference on Robot Learning*, 2024.

[32] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, 2023.

[33] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[34] L. Wang, X. Chen, J. Zhao, and K. He. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. *Advances in Neural Information Processing Systems*, 2024.

[35] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018.

[36] G. Bertasius, H. Wang, and L. Torresani. Is space-time attention all you need for video understanding? In *International Conference on Machine Learning*, 2021.

[37] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.

[38] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, 2020.

A Additional Results

A.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.

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

Table 4: **State-Based Input Results (50 action-labeled trajectories)**

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

Table 5: **State-Based Input Results (100 action-labeled trajectories)**. We report average task success rate for MetaWorld tasks. **Maroon** represents 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.

A.2 CLAM Ablations

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

Table 6: **CLAM Ablation Study State-Based**. We ablate both discrete v.s. continuous actions and with/without joint training. We find that both are critical for achieving a performant policy with both state- and image-based observations. In particular, joint training significantly improves results across all tasks.

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 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 7.

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

Table 7: **Varying Action Decoder Loss Weight**. Increasing the decoder weight improves downstream task performance.

A.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 [21], which provides a dataset of $\sim$ 24 hours of task-agnostic play data. We use trajectories from 10% of the full dataset (13170 trajectories across 30 different tasks) as $\mathcal{D}_{\text{unlabeled}}$ to pretrain CLAM and sample data from five different tasks for $\mathcal{D}_{\text{labeled}}$.

	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**

Table 8: **CALVIN Generalization Results**. CLAM achieves better generalization than VPT on target tasks without any labeled demonstrations for those particular tasks.

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 8 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 Transformer-CLAM. However, the motions of the robot arm in successful rollouts are much more visually similar to the expert trajectories indicating that Transformer-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, Transformer-CLAM achieves more than $2\times$ the success rate of VPT without ever seeing the target task in the action-labeled dataset.

B Avoiding Trivial Solutions for Latent Action Labels

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 previous 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 9 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.

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

Table 9: Image Reconstruction Loss v.s. Cheating Loss

C Detailed Algorithm

Below, we provide a detailed algorithm of training CLAM and latent action policy in Algorithm 1. We also provide the pseudocode for inference rollouts in Algorithm 2.

Algorithm 1 CLAM w/ Joint Action Decoder Training

Input: $\mathcal{D}_{\text{unlabeled}}$, $\mathcal{D}_{\text{labeled}}$, $\mathcal{D}_{\text{unlabeled\_expert}}$, **IDM** f_ϕ, **FDM** g_ψ, **Action Decoder** p_ω, **Latent Action Policy** p_θ
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`
for iter $= 1$ to N_C **do**
  Sample (o_t, o_{t+1}) from $\mathcal{D}_{\text{unlabelled}}$
  $z_t = f_\phi(\cdot \mid o_t, o_{t+1})$ ▷ infer latent action
  $\hat{o}_{t+1} = g_\psi(\cdot \mid o_t, z_t)$ ▷ predict next observation
  $\mathcal{L}_{\text{MSE}}(\phi, \psi) = ||o_{t+1} - \hat{o}_{t+1}||_2^2$ ▷ update IDM and FDM
`# Jointly Train Action Decoder`
  if iter % K == 0 **then**
    Sample $\{(o_t, a_t, o_{t+1})\}_{i=1}^{B}$ from $\mathcal{D}_{\text{labeled}}$ ▷ get batch for action decoder training
    $z_t = f_\phi(\cdot \mid o_t, o_{t+1})$ ▷ infer latent action
    $\hat{a}_t = p_\omega(\cdot \mid z_t)$ ▷ decode latent action
    $\mathcal{L}_{\text{action\_decoder}}(\phi, \omega) = \text{MSE}(\hat{a}_t, a_t)$ ▷ update action decoder and IDM
  end if
end for
`# Stage 2:Train Latent Action Policy`
for iter $= 1$ to N_P **do**
  Sample N_D demos from $\mathcal{D}_{\text{unlabeled\_expert}}$, $\{\tau_i\}_{i=0}^{N_D}$
  $\tau_i^* = \{o_1, f_\phi(o_1, o_2), o_2, \ldots, f_\phi(o_{T-1}, o_T), o_T\}$ ▷ annotate transitions with latent actions
  $\mathcal{L}_{\text{MSE}}(\theta) = ||\hat{z}_{1:T} - z_{1:T}||_2^2$ ▷ update latent policy
end for

Algorithm 2 Inference Time Rollout

Input: **Action Decoder** p_ω, **Latent Policy** π_θ
$o_t = \text{env.reset}()$
while not done **do**
  $z_t = \pi_\theta(\cdot \mid o_t)$ ▷ infer latent action
  $a_t = p_\omega(\cdot \mid z_t)$ ▷ decode latent action
  $o_t, \text{done} = \text{env.step}(a_t)$
end while

D Hyperparameters

We provide detailed hyperparameters for training baseline methods and CLAM.

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 10: BC 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 11: CLAM Pretraining 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 12: 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 13: CALVIN Transformer CLAM Model 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 14: MetaWorld 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 15: CALVIN Environment Hyperparameters

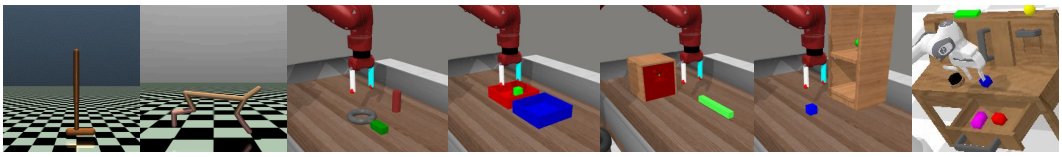

Figure 5: **Evaluation environments in simulation.** We evaluate our approach on both ***locomotion*** tasks from the DMControl benchmark (Hopper and HalfCheetah) and ***manipulation*** tasks (Assembly, Bin Picking, Peg Insert Side, and Shelf Place) from the MetaWorld benchmark. We also evaluate in CALVIN with the Close Drawer and Slider Left tasks.

E Simulation Environments and Tasks

We benchmark our method on DMControl [19], Meta-World [20], and CALVIN [21] without modification. All domains are continuous control environments and we use a fixed episode length and no termination conditions. Figure 5 provides illustrations of each task we evaluate on.

For DMControl tasks, we report normalized return following [22] which normalizes the scores for each environment against an expert score between 0 and 1: 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 [19]: 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 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 [20]. 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.

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

	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

F Real Robot Experimental Setup

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 2. 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 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.

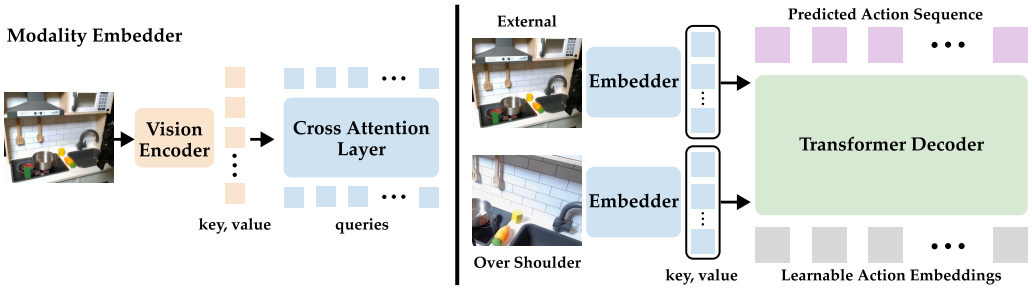

Figure 6: **Real Robot BC Policy.** (Left) Learnable image embeddings following [34]. (Right) The learned image embeddings for each modality are concatenated and provided to a transformer decoder similar to [32]. We also perform action chunking with a chunk size of 5 timesteps for 1 second of execution.

Robot Policy. For our policy, we are inspired by the architectural components introduced in Wang et al. [34] and Zhao et al. [32]. A diagram of our policy architecture is shown in Figure 6. For both external and over-the-shoulder RGB images, we use a pretrained ResNet to first extract $\times 7 \times 7$ 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 [32]. 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 [32] 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 7 shows example real-world task rollouts for the learned policies.

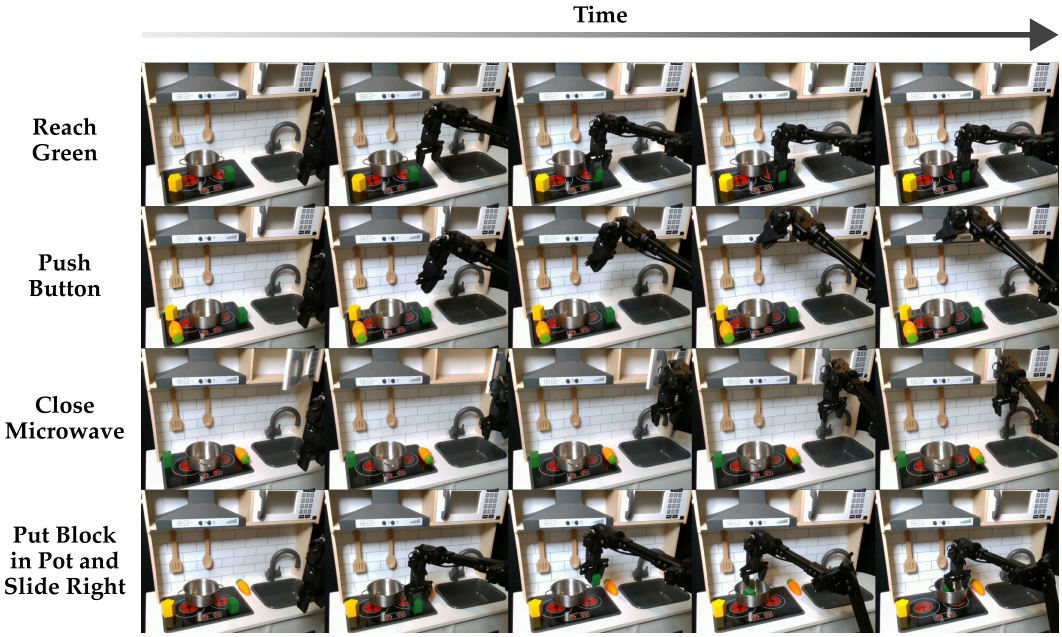

Figure 7: **Real Robot Evaluation Rollouts**. We evaluate CLAM on four manipulation tasks. Each row is an example evaluation rollout and we subsample representative frames interpolated between the first and last timestep.

G Model 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.

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)

G.1 Behavior Cloning Policy

We use an MLP for our latent action policy in simulation and [33] in our real robot experiments. We use the same policy architecture across all the baseline methods to keep the results comparable. 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 [35] 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, $[-1, 1]$.

```
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()
)
```

G.2 Space Time CLAM

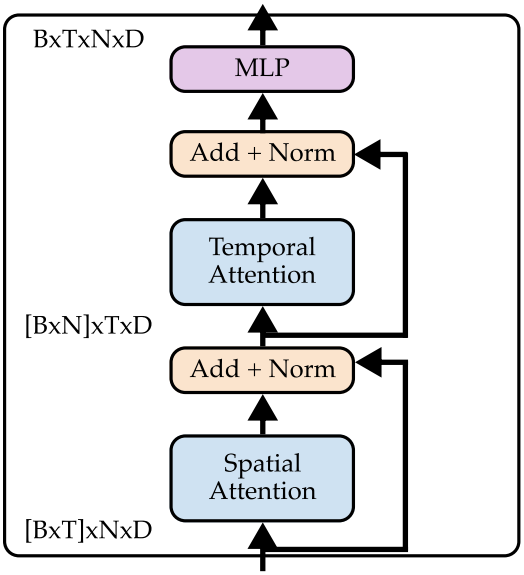

Figure 8: **Space-Time Attention Encoder Block** Model architecture for the Space-Time Attention Encoder Block.

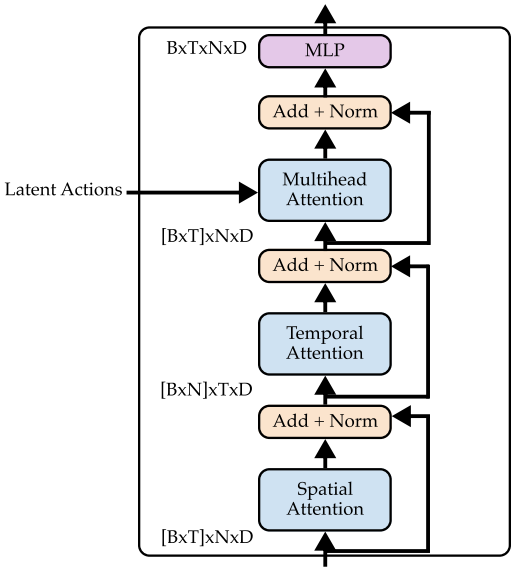

Figure 9: **Space-Time Attention Decoder Block** Model architecture for the Space-Time Attention Decoder Block. Additional multihead attention to condition the learned representation on the latent action outputs from the IDM.

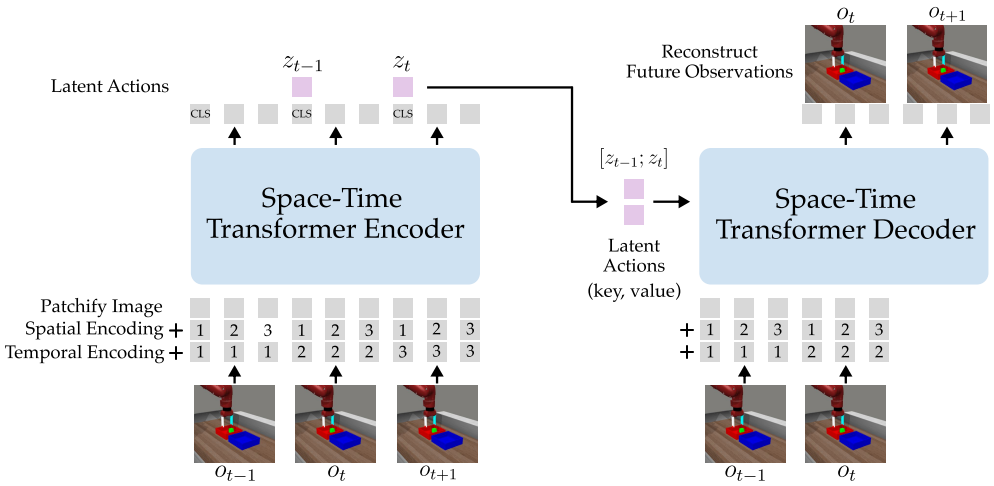

Figure 10: **Space-Time CLAM.** Model architecture for the Space-Time Latent Action Model.

G.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 $3*\texttt{T}$, (o_{t-1}, o_t, o_{t+1}) and the input to the FDM is $2 * \texttt{T} + \texttt{LA}$, (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.

$$
\begin{aligned}
h_{t-1:t+1} &= \text{TransformerEncoder}([o_{t-1}, o_t, o_{t+1}]) \\
z_{t-1:t} &= \text{LatentActionHead}(h_{t:t+1}) \\
o_{t:t+1} &= \text{TransformerDecoder}(h_{t-1:t}, z_{t-1:t})
\end{aligned}
$$

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 [36]. 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 8 and Figure 9. 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 10 summarizes the full architecture diagram for our Space-Time CLAM.

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

```
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.

```
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

```
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)
)
```

H MetaWorld Dataset Generation

TD-MPC [23] is a model-based reinforcement learning (RL) algorithm that performs local trajectory optimization in the latent space of a learned world model. TD-MPC2 [37] 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 11. 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.

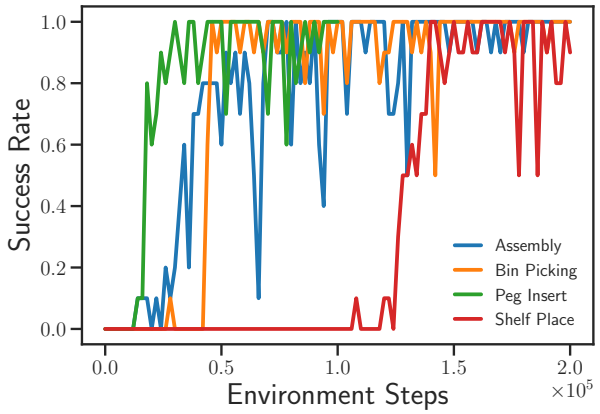

Figure 11: **Task Success Rate for Single-Task TD-MPC2 Agents.** We train single-task agents using TD-MPC2 [37] to generate an offline dataset with different levels of behavior. All four agents exhibit expert performance on their respective tasks by the end of training.

I 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
- 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 [38] 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