REGENT: A Retrieval-Augmented Generalist Agent That Can Act In-Context in New Environments

Kaustubh Sridhar[1], **Souradeep Dutta**[1,2], **Dinesh Jayaraman**[1], **Insup Lee**[1]
[1]University of Pennsylvania, [2]University of British Columbia
ksridhar@seas.upenn.edu

Abstract

Do generalist agents only require large models pre-trained on massive amounts of data to rapidly adapt to new environments? We propose a novel approach to pre-train relatively small models and adapt them to unseen environments via in-context learning, without any finetuning. Our key idea is that retrieval offers a powerful bias for fast adaptation. Indeed, we demonstrate that even a simple retrieval-based 1-nearest neighbor agent offers a surprisingly strong baseline for today's state-of-the-art generalist agents. From this starting point, we construct a semi-parametric agent, REGENT, that trains a transformer-based policy on sequences of queries and retrieved neighbors. REGENT can generalize to unseen robotics and game-playing environments via retrieval augmentation and in-context learning, achieving this with up to 3x fewer parameters and up to an order-of-magnitude fewer pre-training datapoints, significantly outperforming today's state-of-the-art generalist agents. [1]

1 Introduction

AI agents, both in the digital [1, 2, 3, 4, 5] and real world [6, 7, 8, 9, 10, 11], constantly face changing environments that require rapid or even instantaneous adaptation. True generalist agents must not only be capable of performing well on large numbers of training environments, but arguably more importantly, they must be capable of adapting rapidly to new environments. While this goal has been of considerable interest to the reinforcement learning research community, it has proven elusive. The most promising results so far have all been attributed to large models [1, 2, 3, 4, 6], pre-trained on large datasets across many environments, and even these models still struggle to generalize to unseen environments without many new environment-specific demonstrations.

In this work, we take a different approach to the problem of constructing such generalist agents. We start by asking: *Do generalist agents only require large models and massive datasets, or could the right biases help achieve more with less?* Observing that retrieval offers a powerful bias for fast adaptation, we first evaluate a simple 1-nearest neighbor method: "Retrieve and Play (R&P)". To determine the action at the current state, R&P simply retrieves the closest state from a few demonstrations in the target environment and plays its corresponding action. Tested on a wide range of environments, both robotics and game-playing, R&P performs on-par or better than the state-of-the-art generalist agents. Note that these results involve *no pre-training environments*, and not even a neural network policy: it is clear that larger model and pre-training dataset sizes are not the only roadblock to developing generalist agents.

Having thus established the utility of retrieval for fast adaptation of sequential decision making agents, we proceed to incorporate it into the design of a "Retrieval-Augmented Agent" (REGENT). REGENT is a semi-parametric architecture: it pre-trains a transformer policy whose inputs are not only the current state and previous reward, but also retrieved tuples of (state, previous reward, action) from a set of demonstrations for each pre-training task, drawing inspiration from the recent successes

[1] Website: https://kaustubhsridhar.github.io/regent-research/

38th Conference on Neural Information Processing Systems (NeurIPS 2024): Workshop on Open World Agents.

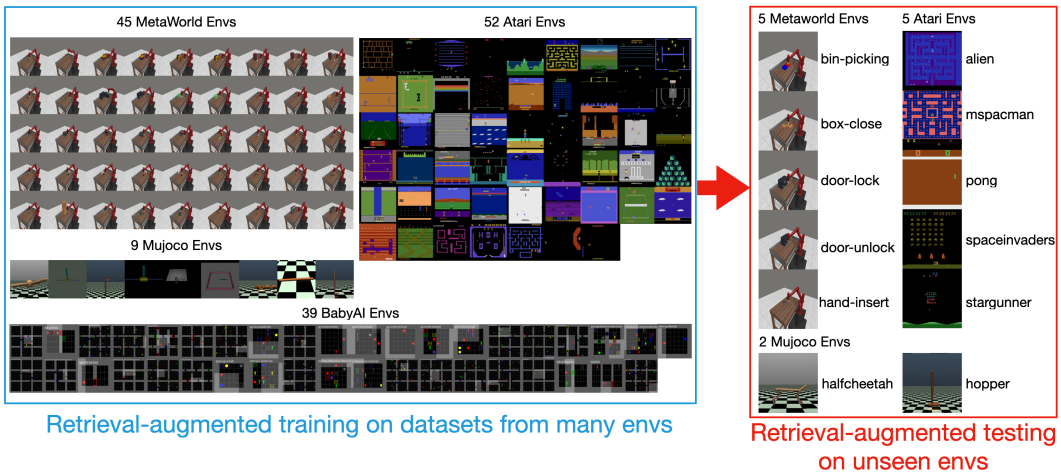

Figure 1: Problem setting in JAT/Gato environments.

of retrieval augmentation in language modeling [12]. At each "query" state, REGENT is trained to prescribe an action through aggregating the action predictions of R&P and the transformer policy. By exploiting retrieval-augmentation as well as in-context learning, REGENT permits near-instantaneous deployment in entirely unseen environments and tasks with only a few demonstrations. REGENT is only one of two models developed so far that can adapt to new environments via in-context learning: the other model is the multi-trajectory transformer (MTT) [3].

We train and evaluate REGENT on two problem settings in this paper, shown in Figures 1 and 2. The first setting is based on the environments used in Gato [1] (and its open source reproduction JAT [2]) and the second setting is based on the ProcGen environments used in MTT [3].

In both settings, REGENT demonstrates significant generalization to unseen environments without any finetuning. In the JAT/Gato setting, REGENT outperforms JAT/Gato even when the baseline is finetuned on demonstrations from the unseen environments. In the ProcGen setting, REGENT significantly surpasses MTT. Moreover, in both settings, REGENT trains a smaller model with 1.4x to 3x fewer parameters and with an order-of-magnitude fewer pre-training datapoints. Finally, while REGENT's design choices are aimed at generalization, its gains are not limited to unseen environments: it even performs better than baselines when deployed within the pre-training environments.

2 Related Work

Recent work in the reinforcement learning community has been aimed at building foundation models and multi-task generalist agents [1, 2, 6, 4, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21].

Many existing generalist agents struggle to adapt to new environments. Gato [1], a popular generalist agent trained on a variety of gameplay and robotic environments, struggles to achieve transfer to an unseen Atari game even after fine-tuning, irrespective of the pretraining data. The authors attribute this difficulty to the "pronounced differences in the visuals, controls, and strategy" among Atari games. They also attribute Gato's lack of in-context adaptation to the limited context length of the transformer not allowing for adequate data to be added in the context. Our method sidesteps this issue by retrieving only limited but relevant parts of demonstration trajectories to include in the context. JAT [2], an open-source version of Gato, faces similar problems. **We compare with and significantly outperform JAT/Gato using fewer parameters and an order-of-magnitude fewer pre-training datapoints.** While REGENT is a 138.6M parameter model, JAT uses 192.7M parameters. The closed-source Gato, with similar performance as the open-source JAT, reports using 1.2B parameters. JAT is also pre-trained on up to 5-10x the amount of data used by our method and yet cannot generalize to unseen environments. Even after finetuning on a few demonstrations from an unseen environment, JAT fails to meaningfully improve.

Many recent generalist agents cannot leverage in-context learning. In-context learning capabilities enable easier and faster adaptation compared to finetuning. Robocat [6], which builds on the Gato model, undergoes many cycles of fine-tuning, data collection, and pre-training from scratch to adapt to new manipulation tasks. The multi-game decision transformer [4], an agent trained on

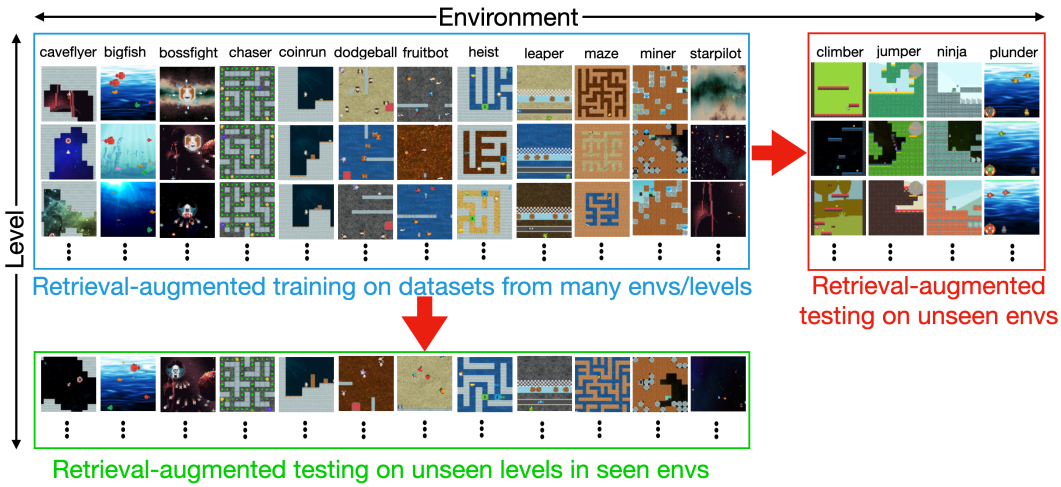

Figure 2: Problem setting in ProcGen environments adapted from [3].

over 50 million Atari gameplay transitions, requires another 1 million transitions for fine-tuning on a held-out game which is not practical in real robot settings. We, on the other hand, show that REGENT (and even R&P) can adapt to new Atari games with as little as 10k transitions and no finetuning on said transitions. Finally, the RT series of robotics models [7, 8, 9], recent Vision-Language-Action models like Octo, OpenVLA, Mobility VLA [10, 11, 13], and other generalist agents like BAKU, RoboAgent [14, 15] do not evaluate or are demonstrated to not possess in-context learning capabilities. REGENT on the other hand can adapt simply with in-context learning to unseen environments.

Agents that can adapt to new tasks via in-context learning, only do so within the same environment. Algorithm Distillation [22], Decision Pretrained Transformer [23], and Prompt Decision Transformer [24] are three in-context reinforcement learning methods proposed to generalize to new goals and tasks within the same environment. None of these can handle changes in visual observations, available controls, and game dynamics.

We also compare with and outperform MTT [3], the only other model that can adapt in-context in the ProcGen setting, with improved data efficiency and a smaller model size. MTT trains a transformer on sequences of trajectories from a particular level and environment and adapts to unseen environments by throwing the demonstrations into its context. The ProcGen variant of REGENT use an order-of-magnitude fewer transitions in pre-training and is about one-third the size of MTT.

Retrieval-augmented generation for training and deployment is a core part of our policy. Various language models trained with retrieval-augmentation such as the original RAG model [25], RETRO [26], and REALM [27] have demonstrated performance on par with vanilla language models with significantly fewer parameters. Moreover, retrieval-augmented generation [12] with large language models has enabled them to quickly adapt to new or up-to-date data. We hope that our work can enable similar capabilities for decision-making agents.

3 Problem Formulation

We aim to pre-train a generalist agent on datasets obtained from different environments, with the goal of generalizing to new unseen environments. The agent has access to a few expert demonstrations in these new environments. In this work, the agent achieves this through in-context learning without any additional finetuning. We assume that the state and action spaces of unseen environments are known.

We model each environment i as a *Markov Decision Process (MDP)*. The i-th *Markov Decision Process (MDP)* $\mathcal{M}_i$ is a tuple $(\mathcal{S}_i, \mathcal{A}_i, \mathcal{P}_i, \mathcal{R}_i, \gamma_i, \mathcal{I}_i)$, where $\mathcal{S}_i$ is the set of states, $\mathcal{A}_i$ is the set of actions, $\mathcal{P}_i(s'|s,a)$ is the probability of transitioning from state s to s' when taking action a, $\mathcal{R}_i(s,a)$ is the reward obtained in state s upon taking action a, $\gamma_i \in [0,1)$ is the discount factor, and $\mathcal{I}_i$ is the initial state distribution. We assume that the MDP's operate over trajectories with finite length H_i, in an episodic fashion. Given a policy π_i acting on $\mathcal{M}_i$, the expected cumulative reward accrued over the duration of an episode (*i.e.*, expected return) is given by $J(\pi_i) = \mathbb{E}_{\pi_i}[\sum_{t=1}^{H_i} \mathcal{R}_i(s_t, a_t)]$.

We denote the expert demonstration dataset corresponding to the i-th (training or unseen) environment consisting of tuples of (state, previous-reward, action) as $\mathcal{D}_i = \{(s_0, 0, a_0), (s_1, r_0, a_1), \ldots, (s_{N_i}, r_{N_i-1}, a_{N_i})\}$. Let us assume that we have access to K such training environments for i from 1 through K, and unseen environments indexed from $K+1$ through M. We assume that the datasets $\mathcal{D}_i$'s are generated by acting according to an expert policy π_i^* which maximizes $J(\pi_i)$. The training environments ($i \leq K$) have a sizeable dataset of expert demonstrations, while the unseen environments have only a handful of them, i.e $|\mathcal{D}_i| >> |\mathcal{D}_j|$, where $j > K$.

4 REGENT: A Retrieval-Augmented Generalist Agent

Simple nearest neighbor retrieval approaches have a long history in few-shot learning [28, 29, 30, 31, 32]. These works have found that, at small training dataset sizes, while parametric models might struggle to extract any signal without extensive architecture or hyperparameter tuning, nearest neighbor approaches perform about as well as the data can support. Motivated by these prior results in other domains, we first construct such an approach for an agent that can learn directly in an unseen environment with limited expert demonstrations in Section 4.1. Then, we consider how to improve this agent through access to experience in pre-training environments, so that it can transfer some knowledge to novel environments that allows it to adapt even more effectively with limited data in Section 4.2.

4.1 Retrieve and Play (R&P)

This is arguably one of the simplest decision agents that leverages the retrieval toolset for adaptation. Given a state s_t from an environment j, let us assume that it is possible to retrieve the n-nearest states (and their corresponding previous rewards, actions) from $\mathcal{D}_j$. We refer to this as the context $c_t \in \mathcal{C}_j$. The set $\mathcal{C}_j$ is the set of all such contexts in environment j. We also call the state s_t the query state following terminology from retrieval-augmented generation for language modeling [12].

The R&P agent takes the state s_t and context c_t as input, picks the nearest retrieved state s' in c_t, and plays the corresponding action a'. That is, $\pi_{\texttt{R\&P}}(s_t, c_t) = a'$. We describe the retrieval process in detail later in Section 4.2. Clearly, R&P is devoid of any learning components which can transfer capabilities from pre-training to unseen environments.

4.2 Retrieval-Augmented Generalist Agent (REGENT)

To go beyond R&P, we posit that if an agent learns to meaningfully combine relevant context to act in a set of training environments, then this skill should be transferable to novel environments as well. We propose exactly such an agent in REGENT. We provide an overview of REGENT in Figure 3 where we depict the reinforcement learning loop with the retrieval mechanism and the REGENT transformer. In the figure, we also include the retrieved context and query inputs to the transformer and its output interpolation with the R&P action. We describe REGENT in detail below.

REGENT consists of a deep neural network policy π_θ, which takes as input the state s_t, previous reward r_{t-1}, context c_t, and outputs the action directly for continuous environments and the logits over the actions in discrete environments. In the context c_t, the retrieved tuples of (state, previous reward, action) are placed in order of their closesness to the query state s_t with the closest retrieved state s' placed first. Let $d(s_t, s')$ be the distance between s_t and s'. We perform a distance weighted interpolation between the outputs of the neural network policy and R&P as follows,

$$\pi^{\theta}_{\texttt{REGENT}}(s_t, r_{t-1}, c_t) = e^{-\lambda d(s_t, s')}\pi_{\texttt{R\&P}}(s_t, c_t) + (1 - e^{-\lambda d(s_t, s')})\,\sigma(\pi_\theta(s_t, r_{t-1}, c_t)) \tag{1}$$

$$\sigma(x) = \begin{cases} \mathsf{Softmax}(x), & \text{if action space is discrete} \\ L \times \mathsf{MixedReLU}(x), & \text{if action space is continuous} \end{cases} \tag{2}$$

where $\mathsf{MixedReLU} : \mathbb{R} \to [-1, 1]$ is a tanh-like activation function from [33] and is detailed in Appendix A. Further, $L \in \mathbb{R}$ is a hyperparameter for scaling the output of the neural network for continuous action spaces after the MixedReLU. We simply set both L and λ to 10 everywhere following a similar choice in [33]. The function π_θ is a causal transformer, which is adept at modeling relatively long sequences of contextual information and predicting the optimal action [25, 26, 34, 4]. All distances are normalized and clipped to $[0, 1]$ as detailed in Appendix A. For discrete action

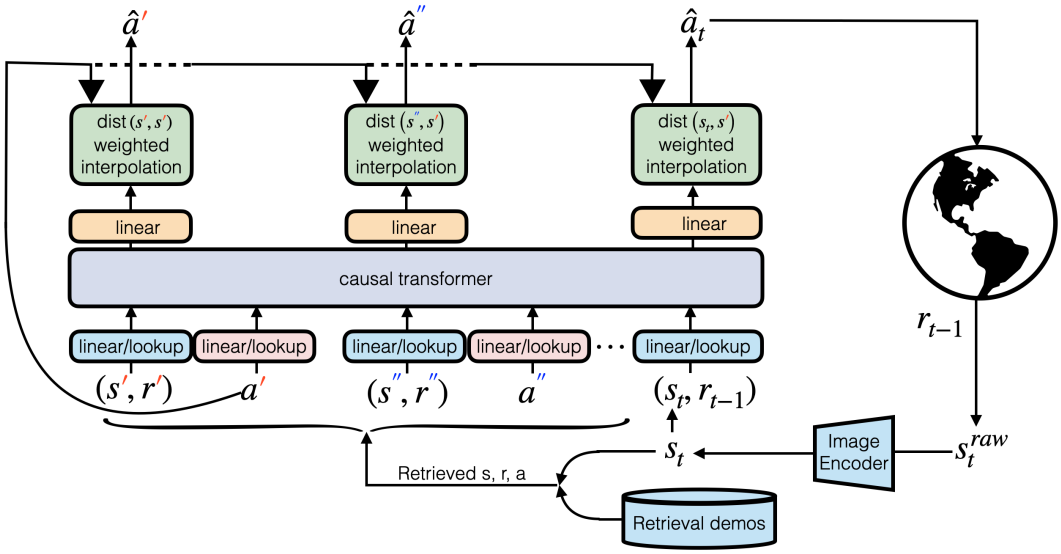

Figure 3: **The REGENT architecture and overview.** (1) A query state (from the unseen environment during deployment or from training environments' datasets during pre-training) is processed for retrieval. (2) The n nearest states from a few demonstrations in an unseen environment or from a designated retrieval subset of pre-training environments' datasets are retrieved. These states, and their corresponding previous rewards and actions, are added to the context in order of their closeness to the query state, followed by the query state and previous reward. (3) The predictions from the REGENT transformer are combined with the first retrieved action. (4) At deployment, only the predicted query action is used. During pre-training, the loss from predicting all actions is used to train the transformer.

spaces, where the transformer outputs a distribution over actions, we modify $\pi_{\texttt{R\&P}}$ as follows. Let us assume that the discrete action space consists of N_{act} different actions.

$$\pi_{\texttt{R\&P}}(a|s_t, c_t) = \begin{cases} \frac{1+(N_{\text{act}}-1)(1-d(s_t,s'))}{N_{\text{act}}} & \text{, if } a = a' \\ \frac{d(s_t,s')}{N_{\text{act}}} & \text{, if } a \neq a' \end{cases} \tag{3}$$

The distribution induced by the modified $\pi_{\texttt{R\&P}}$ function assigns all probability mass to the action a' when $d(s_t, s') = 0$ and acts like a uniform distribution when $d(s_t, s') = 1$. The latter reduces the retrieval bias when $d(s_t, s')$ is large. As shown in Figure 3, for predicting each action, either for a retrieved state in the context or for the query state, we set $d(s, s')$ in Equations (1) and (3) to the distance between that state s and the first (closest) retrieved state in the context s'.

Further, the exponential weights in Equation (1) allows us to smoothly transition between R&P and the parametric policy π_θ. When the state s_t is close enough to first retrieved state in the context c_t, $\pi_{\texttt{REGENT}}$ simply plays the retrieved action. However, as it moves further away, policy $\pi_{\texttt{R\&P}}$ becomes a uniform distribution and the parametric policy π_θ takes more precedence. *We also hypothesize that this interpolation allows the transformer to more readily generalize to unseen environments, since it is given the easier task of predicting the residual to the R&P action rather than predicting the complete action.*

Pre-training REGENT and Loss Function: We train REGENT by minimizing the total cross-entropy loss on discrete actions and total mean-squared error on continuous actions for all $n + 1$ action predictions (n in the context and 1 query). We also follow the JAT recipe in ensuring that each training batch consists only of data from a single environment's dataset. We provide details about the training hyperparameters in Appendix A.

REGENT Architecture: REGENT adapts the JAT architecture in the JAT/Gato setting. It consists of a causal transformer trunk. It has a shared linear encoder for image embeddings, vector observations, and continuous actions. This encoder takes a maximum input vector of length 513, accommodating the largest input vectors—image embeddings of size 512—plus a single reward value. When the length of an input vector is less than that, it is cyclically repeated and padded to the maximum length. It has a large shared lookup table with the GPT2 [35] vocabulary size for encoding discrete actions, discrete observations, and text observations. Another linear layer is used to combine multiple discrete

values and text tokens present in a single observation vector. It has a shared linear head for predicting continuous actions. A shared linear head is used for predicting distributions over discrete actions. All of the above components are shared across environments but only a subset of input encoders and output heads may be triggered during a forward and backward pass depending on the input modalities in an environment and output action space. We note that states and rewards are concatenated together before encoding following the JAT recipe [2]. REGENT has a total of 138.6M parameters, including the frozen ResNet18 image encoder, compared to JAT's 192.7M parameters.

We simplify REGENT for the ProcGen setting keeping the transformer trunk with only a convolution encoder for direct image inputs; a lookup table for encoding discrete actions; and a linear head for predicting distributions over discrete actions. Here, following the MTT recipe, we do not use any rewards. We only use the states and actions. Here, REGENT has a total of 116M parameters compared to MTT's 310M parameters. We detail all architectural hyperparameters and the differences between JAT, MTT, and REGENT in Appendix A.

REGENT **Training Data and Environment Suites**: In the JAT/Gato setting, we pre-train on 100k transitions in each of the 45 Metaworld training environments, 9 Mujoco training environments, 52 Atari training environments, and 39 BabyAI training environments. This adds up to a total of 14.5M transitions used to pre-train REGENT. We obtain these transitions by taking a subset of the open-source JAT dataset [2]. The complete JAT dataset consists of 5-10x the amount of data we use in each environment. The Atari environments have 18 discrete actions and four consecutive grayscale images as states. However, we embed each combined four-stacked image into a vector of 512 dimensions and use this image embedding everywhere (for R&P and REGENT). The details of embedding images are discussed later in this Section. The Metaworld and Mujoco environments have proprioceptive vector observations and continuous action spaces. All Metaworld environments have observations with 39 dimensions and actions with 4 dimensions. Mujoco environments have observations with 4 to 376 dimensions and actions with 1 to 17 dimensions. The BabyAI environments have discrete observations and up to 7 discrete actions. They also have text observations specifying a goal in natural language. Together, after tokenizing the text into 64 tokens, BabyAI observations have 212 dimensions and consist only of discrete values.

In the ProcGen setting that borrows from MTT [3], we generate 20 rollouts with PPO policies on each of the 12 training environments for various levels depending on the environment. We do so to ensure that the total number of transitions in each environment is 1M. The total size of the pre-training dataset is 12M. All environments have an image-based state space and discrete actions. The number of levels varies from 63 at the smallest in bigfish to 1565 at the largest in maze. This variation arises from the difference in rollout horizon in each environment. We note that, even the largest value of 1565 levels in maze is an order-of-magnitude fewer than the 10,000 levels in each game used by MTT for pre-training.

Processing Raw Observations, Distance Metrics, Retrieval Mechanism, and Preprocessing Training Data: In the JAT/Gato setting, if the raw observations consist of images, we embed the image with an off-the-shelf ResNet18 encoder [36] trained only on ImageNet data. If instead, the raw observations are proprioceptive vectors, we use them directly without modification. In the ProcGen setting, the raw observations consist of images that we use directly without modification.

In the JAT/Gato setting, we use the ℓ_2 distance metric to compute distances between pairs of observations, regardless of whether they are image embeddings or proprioceptive vectors. To significantly speed up the search for nearest states to a query state, we leverage similarity search indices [37]. In the ProcGen setting, we utilize the SSIM distance metric [38] to obtain distances between two images. We perform a naive search for the nearest images to a query image, however all SSIM distance calculations are parallelized on GPUs to speed up the retrieval process.

R&P simply performs the retrieval process described above at evaluation time, obtains the closest state to a query state, and plays the corresponding action. REGENT on the other hand has to setup its pre-training dataset as follows. It first designates a certain number of randomly chosen demonstrations per environment as the retrieval set in that environment. This is described in Appendix A. Then, for each state in each environment's training dataset, we retrieve the $n = 19$ closest states from the designated retrieval subset for that dataset. We ensure that none of the retrieved states are from the same demonstration as the query state. In this process, we convert our dataset of transitions to a dataset of (context, query state, query reward) datapoints where the context consists of 19 retrieved (state, reward, action) tuples. Now, we can begin pre-training REGENT on this dataset.

Evaluating REGENT: After pre-training, in each of the two settings, we have one model that can be deployed directly, without finetuning, on unseen environments. During deployment in an environment, at each step, we retrieve the $n = 19$ closest states to the current query state (and their corresponding actions and rewards). We pass this context with the query state, and previous reward into Equation (1) through the architecture described above to predict the query action to take in the environment.

In the JAT/Gato setting, for the unseen environments, we hold-out 5 Metaworld, 5 Atari, and two Mujoco environments (see Figure 1). In the ProcGen setting, following MTT [3], we hold-out 5 environments. Unlike MTT, we also evaluate on unseen levels in training environments (see Figure 2). We explain these choices in Appendix A. In this work, we focus on unseen environments in the same suite as training environments and leave further generalization to future work. Finally, we also note that in all game-playing environments, we add a small sticky probability [39] of 0.05 in unseen Atari environments and 0.2 in unseen ProcGen environments following [3]. This is not present in any data or demonstration, which induces further stochasticity and tests the ability of both R&P and REGENT to truly generalize under novel and stochastic dynamics.

5 Sub-Optimality Bound for REGENT Policies

In this section, we aim to bound the sub-optimality of the REGENT policy. This is measured with respect to the expert policy π^*_j, that generated the retrieval demonstrations $\mathcal{D}_j$. We focus on the discrete action case here and leave the continuous action case for future work. The sub-optimality gap in (training or unseen) environment j is given by $(J(\pi^*_j) - J(\pi^\theta_{\texttt{REGENT}}))$. Inspired by the theoretical analysis of [33], we define the "most isolated state" and use this definition to bound the total variation in the REGENT policy class and hence the sub-optimality gap.

That is, first, given $\mathcal{D}_j$, we wish to obtain the maximum value of the distance term $d(s_t, s')$ in Equations (3), (1). To do so, we define the most isolated state as follows.

Definition 5.1 (Most Isolated State)**.** For a given set of retrieval demonstrations $\mathcal{D}_j$ in environment j, we define the most isolated state $s^I_{\mathcal{D}_j} := \arg\max_{s \in S_j} \big(\min_{s' \in \mathcal{D}_j} d(s, s') \big)$, and consequently the distance to the most isolated state as $d^I_{\mathcal{D}_j} = \min_{s' \in \mathcal{D}_j} d(s^I_{\mathcal{D}_j}, s')$.

All distances between a state in this environment and its closest retrieved state are less than the above value, which also measures state space coverage by the demonstrations available for retrieval. Using the above definition, we have the following.

Theorem 5.2. *The sub-optimality gap in environment j is* $\mathbf{J}(\pi^*_\mathbf{j}) - \mathbf{J}(\pi^\theta_{\textit{REGENT}}) \leq \mathbf{min}\{\mathbf{H}, \mathbf{H^2}(\mathbf{1} - \mathbf{e}^{-\lambda \mathbf{d^I_{\mathcal{D}_j}}})\}$

Proof: We provide the proof in Appendix B.

The main consequence of this theorem, also observed in our results, is that the sub-optimality gap reduces with more demonstrations in $\mathcal{D}_j$ because of the reduced distance to the most isolated state.

6 Experimental Evaluation

In our experiments, we aim to answer the following key questions in the two settings depicted in Figures 1 and 2. (1) How well can R&P and REGENT generalize to unseen environments? (2) How does finetuning in the new environments improve REGENT? (3) How well can REGENT generalize to variations of the training environments and perform in aggregate on training environments? (4) How does REGENT qualitatively compare with R&P in using the retrieved context?

Metrics: We plot the normalized return computed using the return of a random and expert agent in each environment as $\frac{(\text{total return} - \text{random return})}{(\text{expert return} - \text{random return})}$. The expert and random returns for all JAT/Gato environments are made available in the original work [2]. The expert and random returns for the ProcGen environments can be found in the original ProcGen paper [40].

Baselines: We consider a different set of baselines for each of the two settings. In the JAT/Gato setting, we compare R&P and REGENT with two variants of JAT. The first is a JAT model trained on the same dataset as REGENT. The second is a JAT model trained on all available JAT data which consists of an order of magnitude more datapoints (in particular, 5x more data in Atari environments and 10x

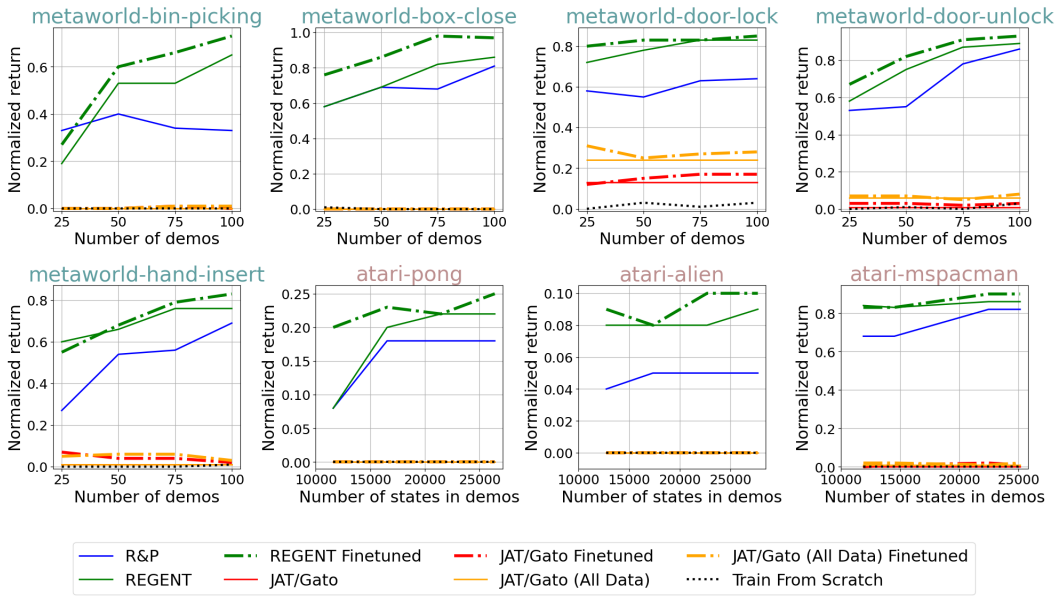

Figure 4: Normalized returns in the unseen Metaworld and Atari environments against the number of demonstration trajectories the agent can retrieve from or finetune on. Each value is an average across 100 rollouts of different seeds in Metaworld and 15 rollouts of different seeds (with $p_{\text{sticky}} = 0.05$) in Atari. See Table 1 for detailed results.

more data in all other environments). We label the former apples-to-apples comparison as `JAT/Gato` and the latter as `JAT/Gato (All Data)`. All hyperparameters are in Appendix A.

In the ProcGen setting, we compare with MTT [3]. MTT only reports results on unseen environments and not on unseen levels in training environments. We simply take the best MTT result on each of the four unseen environments in [3], obtained when 4 demonstrations are included in the MTT context.

Finetuning and Train-from-scratch Baselines: We finetune `JAT/Gato` and `JAT/Gato (All Data)` on various number of demonstrations in each unseen environment in the `JAT/Gato` setting and compare with `REGENT`. We also compare with a train-from-scratch behavior cloning baseline. For completeness, we also finetune `REGENT` on various number of demonstrations in each unseen environment in the `JAT/Gato` setting. The finetuning hyperparameters are the same for all methods and can be found with the train-from-scratch details in Appendix A.

Generalization to Unseen Environments: We plot the normalized return obtained by all methods in unseen Metaworld and Atari environments from the `JAT/Gato` setting for various number of demonstrations (25, 50, 75, 100) in Figure 4. For a fair comparison across Atari environments with marked differences in episode horizons, we plot normalized return against various number of states in the demonstrations (atleast 10k, 15k, 20k, 25k) rather than the number of demonstrations itself. These demonstrations can be used by the different methods for retrieval or fine-tuning. We also plot the normalized return obtained by all methods in unseen ProcGen environments against various number of demonstrations (2, 4, 8, 12, 16, 20) to retrieve from in Figure 5. As mentioned before in Section 4.2, unseen Atari environments have a sticky probability of 0.05 and unseen ProcGen environments have a sticky probability of 0.2.

In both Figures 4 and 5, we observe that `R&P` and `REGENT` can generalize well to unseen Atari and ProcGen environments with image observations and discrete actions as well as to unseen Metaworld environments with vector observations and continuous actions. `R&P` is a surprisingly strong baseline, but `REGENT` improves on `R&P` consistently. In general, both methods appear to steadily improve with more demonstrations (with only a few exceptions).

In Figure 4, we observe that `JAT/Gato` cannot generalize to most unseen environments. `REGENT` (and even `R&P`) outperform even the `All Data` variants of `JAT/Gato` which were pre-trained on 5-10x the size of the `REGENT` dataset. The sticky probability in unseen Atari environments adds further stochasticity and stress tests true generalization against simply replaying demonstrations.

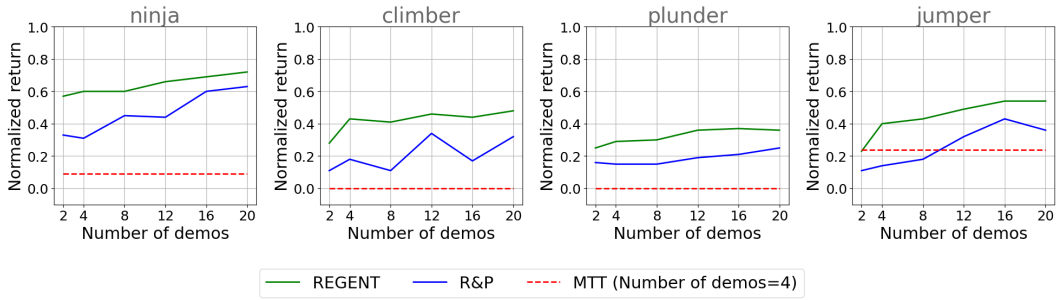

Figure 5: Normalized returns in unseen ProcGen environments against the number of demonstration trajectories the agent can retrieve from. Each value is an average across 10 levels with 5 rollouts each with $p_{\text{sticky}} = 0.2$. See Table 7 for detailed results.

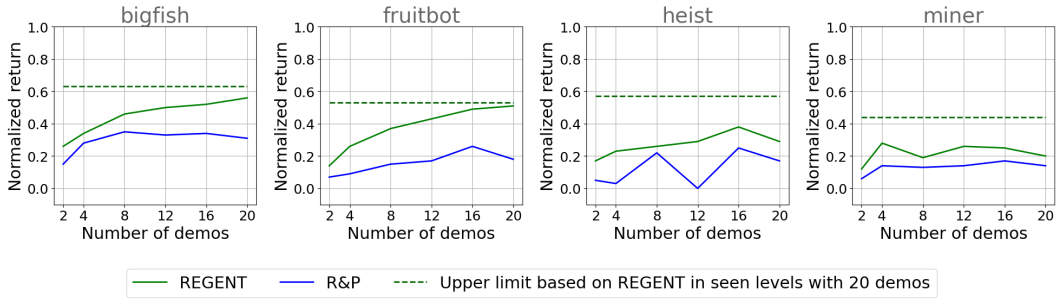

Figure 6: Normalized returns in unseen levels in 4 (of 12) ProcGen training environments against the number of demonstration trajectories the agent can retrieve from. Each value is an average across 10 levels with 5 rollouts each with $p_{\text{sticky}} = 0.1$. The other 8 plots are similar to the above and can be found in Figure 15 (in Appendix D). See Table 8 for detailed results.

In Figure 5, `REGENT` (and even `R&P`) outperform MTT which was trained on an order of magnitude more training levels and data than `REGENT` while also having approximately 3x the parameters.

Effect of Finetuning: From Figure 4, we see that `JAT/Gato`, along with its `All Data` variant, struggles to perform even after finetuning. Training from scratch on the few demonstrations in each unseen environment also does not obtain any meaningful performance in most environments. `JAT/Gato` (and even `R&P`), without any finetuning, outperform both finetuned variants of `JAT/Gato`. Moreover, we can see that `REGENT` further improves marginally after finetuning, even with only a few demonstrations.

To highlight `REGENT`'s ability to adapt from very little data, we note that in Figure 4, we only vary the number of states in Atari demonstrations until 25k. Whereas, the closest generalist policy that finetunes to new Atari environments, the multi-game decision transformer [4], requires 1M transitions. Finally, we note that all methods face a hurdle in generalizing to the very long-horizon atari-spaceinvaders and atari-stargunner environments, which have horizons about 10x that of atari-pong and hence have not been shown in Figure 4. Generalization to the new embodiments in the two unseen Mujoco environments proves equally challenging, we discuss this in detail in Appendix C.

Generalization to Variations of the Training Environments: In the ProcGen setting in Figure 2, the unseen levels of the 12 training environments provide an avenue to test a middle ground of generalization between training and unseen environments. We plot the normalized returns on unseen levels of 4 (of 12) training environments (with a sticky probability of 0.1) in Figure 6. The remaining can be found in Figure 15 (Appendix D) and are similar to the above 4. In Figure 6, we again observe that `REGENT` performs the best in unseen levels while `R&P` remains a strong baseline. We also depict the performance of `REGENT` on seen levels of these training environments with a dotted line in this Figure. This represents an upper bound for `REGENT`'s performance in unseen levels. We observe that with a large number of demonstrations, `REGENT` appears to reach close to this upper bound simply via retrieval-augmentation and in-context learning in some environments.

Aggregate Performance on Training Environments: We plot the aggregate normalized return, both IQM [41] and mean, on training environments for each of the 4 suites (in the `JAT/Gato` setting) in Figure 7. We notice that `REGENT` significantly exceeds `JAT/Gato` on Metaworld, exceeds it on Atari,

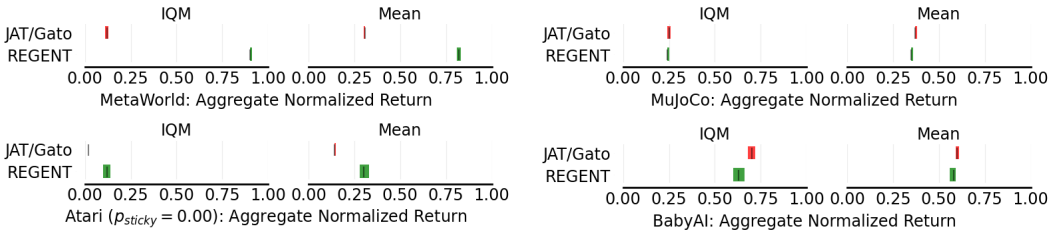

Figure 7: Aggregate normalized returns in the 45 training Metaworld environments, 9 training Mujoco environments, 52 training Atari environments, and 39 training BabyAI environments. See Figures 11, 12, 13, 14 (in Appendix C) for performance on each training environment of each suite.

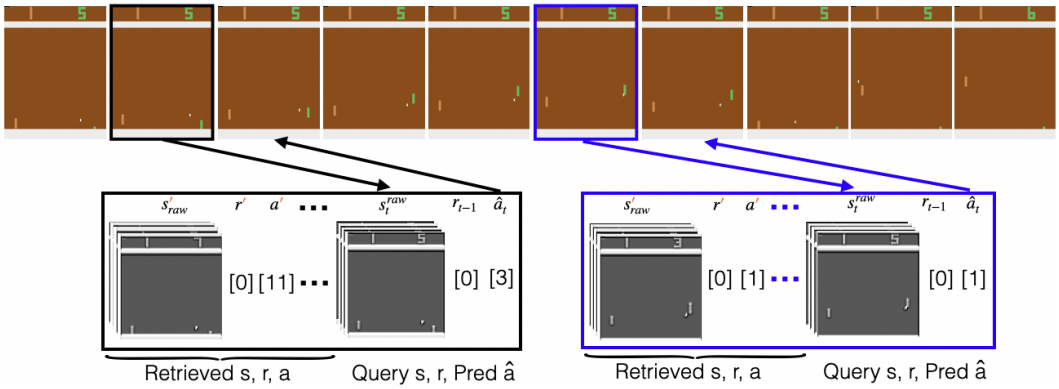

Figure 8: Qualitative examples of a few inputs and outputs of REGENT for two states in a rendered rollout in the unseen, discrete action space, atari-pong environment. REGENT leverages its in-context learning capabilities and interpolation with R&P to either make a simple decision and predict the same action as R&P (see blue box on the right) or predict better actions at key states (see black box on the left) that leads to better overall performance as seen in Figure 4.

matches it on Mujoco, and is close to matching it on BabyAI. This demonstrates the dual advantage of REGENT which gains the capability to generalize to unseen environments while preserving overall multi-task performance in training environments.

Qualitative Examples: We plot examples of the inputs and outputs of REGENT at various states during a rendered rollout in the atari-pong environment in Figure 8 to highlight REGENT's learned in-context learning capabilities and interpolation with R&P. We also provide a qualitative example in a continuous metaworld environment in Appendix C.

7 Conclusions and Future Work

In this work, we demonstrated that retrieval offers generalist agents a powerful bias for rapid adaptation to new environments, even without large models and vast datasets. We showed that a simple retrieval-based 1-nearest neighbor agent, Retrieve and Play (R&P), is a strong baseline that matches or exceeds the performance of today's state-of-the-art generalist agents in unseen environments. Building on this, we proposed a semi-parametric generalist agent, REGENT, that pre-trains a transformer-based policy on sequences of query state, reward, and retrieved context. REGENT exploits retrieval-augmentation and in-context learning for near-instantaneous deployment in unseen environments with only a few demonstrations. Even after pre-training on an order of magnitude fewer datapoints than other generalist agents and with fewer parameters, REGENT outperforms them across unseen environments and surpasses them even after they have been finetuned on demonstrations from those unseen environments. REGENT, itself, further improves with finetuning on even a small number of demonstrations. We note that REGENT faces a couple of limitations: adapting to new embodiments and long-horizon environments. In future work, we believe a larger diversity of embodiments in the training dataset and improved retrieval from longer demonstrations can help REGENT overcome these challenges. We conclude with the conviction that retrieval in general and REGENT in particular redefines the possibilities for developing highly adaptive and efficient generalist agents, even with limited resources.

Acknowledgements: This work was supported in part by ARO MURI W911NF-20-1-0080, NSF 2143274, NSF CAREER 2239301, NSF 2331783, and ONR N00014-22-12677. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views the Army Research Office (ARO), Office of Naval Research (ONR), the Department of Defense, or the United States Government.

References

[1] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. (Cited on 1, 2)

[2] Quentin Gallouédec, Edward Beeching, Clément Romac, and Emmanuel Dellandréa. Jack of all trades, master of some, a multi-purpose transformer agent. *arXiv preprint arXiv:2402.09844*, 2024. (Cited on 1, 2, 6, 7, 15)

[3] Sharath Chandra Raparthy, Eric Hambro, Robert Kirk, Mikael Henaff, and Roberta Raileanu. Generalization to new sequential decision making tasks with in-context learning. *arXiv preprint arXiv:2312.03801*, 2023. (Cited on 1, 2, 3, 6, 7, 8, 16, 25)

[4] Kuang-Huei Lee, Ofir Nachum, Mengjiao Sherry Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35:27921–27936, 2022. (Cited on 1, 2, 4, 9, 16)

[5] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. (Cited on 1)

[6] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023. (Cited on 1, 2)

[7] 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. (Cited on 1, 2, 3)

[8] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023. (Cited on 1, 2, 3)

[9] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023. (Cited on 1, 2, 3)

[10] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024. (Cited on 1, 2, 3)

[11] 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. (Cited on 1, 2, 3)

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023. (Cited on 2, 3, 4)

[13] Hao-Tien Lewis Chiang, Zhuo Xu, Zipeng Fu, Mithun George Jacob, Tingnan Zhang, Tsang-Wei Edward Lee, Wenhao Yu, Connor Schenck, David Rendleman, Dhruv Shah, et al. Mobility vla: Multimodal instruction navigation with long-context vlms and topological graphs. *arXiv preprint arXiv:2407.07775*, 2024. (Cited on 2, 3)

[14] Siddhant Haldar, Zhuoran Peng, and Lerrel Pinto. Baku: An efficient transformer for multi-task policy learning. *arXiv preprint arXiv:2406.07539*, 2024. (Cited on 2, 3)

[15] 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. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4788–4795. IEEE, 2024. (Cited on 2, 3)

[16] Kiana Ehsani, Tanmay Gupta, Rose Hendrix, Jordi Salvador, Luca Weihs, Kuo-Hao Zeng, Kunal Pratap Singh, Yejin Kim, Winson Han, Alvaro Herrasti, et al. Imitating shortest paths in simulation enables effective navigation and manipulation in the real world. *arXiv preprint arXiv:2312.02976*, 2023. (Cited on 2)

[17] Jonathan Yang, Dorsa Sadigh, and Chelsea Finn. Polybot: Training one policy across robots while embracing variability. *arXiv preprint arXiv:2307.03719*, 2023. (Cited on 2)

[18] Dhruv Shah, Ajay Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. Gnm: A general navigation model to drive any robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7226–7233. IEEE, 2023. (Cited on 2)

[19] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. Vint: A foundation model for visual navigation. *arXiv preprint arXiv:2306.14846*, 2023. (Cited on 2)

[20] Jonathan Yang, Catherine Glossop, Arjun Bhorkar, Dhruv Shah, Quan Vuong, Chelsea Finn, Dorsa Sadigh, and Sergey Levine. Pushing the limits of cross-embodiment learning for manipulation and navigation. *arXiv preprint arXiv:2402.19432*, 2024. (Cited on 2)

[21] Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024. (Cited on 2)

[22] Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022. (Cited on 3)

[23] Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on 3)

[24] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *international conference on machine learning*, pages 24631–24645. PMLR, 2022. (Cited on 3)

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. (Cited on 3, 4)

[26] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022. (Cited on 3, 4)

[27] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020. (Cited on 3)

[28] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens Van Der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019. (Cited on 4)

[29] M Saiful Bari, Batool Haider, and Saab Mansour. Nearest neighbour few-shot learning for cross-lingual classification. *arXiv preprint arXiv:2109.02221*, 2021. (Cited on 4)

[30] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016. (Cited on 4)

[31] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017. (Cited on 4)

[32] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019. (Cited on 4)

[33] Kaustubh Sridhar, Souradeep Dutta, Dinesh Jayaraman, James Weimer, and Insup Lee. Memory-consistent neural networks for imitation learning. *arXiv preprint arXiv:2310.06171*, 2023. (Cited on 4, 7, 15, 16)

[34] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021. (Cited on 4)

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. (Cited on 5)

[36] Sudeep Dasari, Mohan Kumar Srirama, Unnat Jain, and Abhinav Gupta. An unbiased look at datasets for visuo-motor pre-training. In *Conference on Robot Learning*, pages 1183–1198. PMLR, 2023. (Cited on 6, 15)

[37] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024. (Cited on 6)

[38] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. (Cited on 6)

[39] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. (Cited on 7)

[40] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020. (Cited on 7)

[41] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021. (Cited on 9)

[42] Aaditya Singh, Stephanie Chan, Ted Moskovitz, Erin Grant, Andrew Saxe, and Felix Hill. The transient nature of emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on 15)

[43] 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*, pages 1407–1416. PMLR, 2018. (Cited on 15)

[44] Kaustubh Sridhar, Oleg Sokolsky, Insup Lee, and James Weimer. Improving neural network robustness via persistency of excitation. In *2022 American Control Conference (ACC)*, pages 1521–1526. IEEE, 2022. (Cited on 16)

[45] Yahan Yang, Ramneet Kaur, Souradeep Dutta, and Insup Lee. Interpretable detection of distribution shifts in learning enabled cyber-physical systems. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 225–235. IEEE, 2022. (Cited on 16)

[46] Kaustubh Sridhar, Souradeep Dutta, James Weimer, and Insup Lee. Guaranteed conformance of neurosymbolic models to natural constraints. In *Learning for Dynamics and Control Conference*, pages 76–89. PMLR, 2023. (Cited on 16)

[47] Jean Park, Sydney Pugh, Kaustubh Sridhar, Mengyu Liu, Navish Yarna, Ramneet Kaur, Souradeep Dutta, Elena Bernardis, Oleg Sokolsky, and Insup Lee. Automating weak label generation for data programming with clinicians in the loop. In *2024 IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 25–36. IEEE, 2024. (Cited on 16)

[48] Souradeep Dutta, Kaustubh Sridhar, Osbert Bastani, Edgar Dobriban, James Weimer, Insup Lee, and Julia Parish-Morris. Exploring with sticky mittens: Reinforcement learning with expert interventions via option templates. In *Conference on Robot Learning*, pages 1499–1509. PMLR, 2023. (Cited on 16)

[49] 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*, pages 1094–1100. PMLR, 2020. (Cited on 16)

[50] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020. (Cited on 16)

[51] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019. (Cited on 16)

[52] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. (Cited on 16)

[53] Kaustubh Sridhar and Srikant Sukumar. Finite-time, event-triggered tracking control of quadrotors. In *5th CEAS Specialist Conference on Guidance, Navigation & Control (EurGNC 19) Milano, Italy*, 2019. (Cited on 16)

[54] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The uva/padova type 1 diabetes simulator: new features. *Journal of diabetes science and technology*, 8(1):26–34, 2014. (Cited on 16)

[55] Nived Rajaraman, Lin F. Yang, Jiantao Jiao, and Kannan Ramchandran. Toward the fundamental limits of imitation learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. (Cited on 17)

Appendix

A Additional Details on REGENT: A Retrieval Augmented Generalist Agent

MixedReLU activation: MixedReLU is a tanh-like activation function from [33] given by $\mathsf{MixedReLU}(x) = (2(\mathsf{ReLU}(\frac{x+1}{2}) - \mathsf{ReLU}(\frac{x-1}{2})) - 1)$ which simplifies to -1 for $x < -1$, x for $-1 \leq x \leq 1$, and 1 for $x > 1$.

Normalizing distances to $[0, 1]$: We compute the 95th percentile of all distances $d(s, s')$ between any (retrieved or query) state s and the first (closest) retrieved state s'. This value is computed from the demonstrations $\mathcal{D}_j$ and is used to normalize all distances in that environment. This value is calculated for all (training or unseen) environments during the preprocessing stage. If after normalization, a distance value is greater than 1, we simply clip it to 1.

Architectural hyperparameters: In both settings, the transformer trunk consists of 12 layers and 12 heads with a hidden size of 768. We set the maximum position encodings to 40 for 20 (state, previous reward)'s and 20 actions. Of these 20, 19 belong to the retrieved context and 1 belongs to the query. In the JAT/Gato setting, the maximum multi-discrete observation size is 212 (for BabyAI). The maximum continuous observation size as discussed before in Section 4 is set to 513, a consequence of the ResNet18 image embedding models's [36] embedding size of 512 with an additional 1 dimension for the reward. All linear encoding layers map from their corresponding input size to the hidden size. Following the JAT model, the linear decoder head for predicting continuous actions maps from the hidden size to the maximum continuous size discussed above (513). On the other hand, the linear decoder head for predicting distributions over discrete actions maps from the hidden size to only the maximum number of discrete actions across all discrete environments (18), whereas in the JAT model, they map to the full GPT2 vocab size of 50257.

In the JAT/Gato setting, the original JAT architecture has a much larger context size (that is not required for REGENT), a larger image encoder (than the resnet18 used in REGENT), a much larger discrete decoder predicting distributions over the entire GPT2 vocab size (instead of just the maximum number of discrete actions in REGENT), and has (optional) decoders for predicting various observations (that is also not required for REGENT).

In the ProcGen setting, MTT has a 18 layers, 16 heads, and a hidden size of 1024 – all three of which are much larger than REGENT's architectural hyperparameters mentioned above.

Training hyperparameters: In the JAT/Gato setting, we use a batch size of 512 and the AdamW optimizer with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate starts at 5e-5 and decays to zero through 3 epochs over all pre-training data. We early stop after a single epoch because we find that overtraining reduces in-context learning performance. This observation is consistent with similar observations about the transience of in-context learning in literature [42].

In the ProcGen setting, we use a batch size of 1024, a starting learning of 1e-4 also decaying over 3 epochs over all the pre-training data with an early stop after the first epoch. We again use the AdamW optimizer with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.95$.

Re-training JAT/Gato hyperparameters: When retraining JAT/Gato, we follow all hypepermaters chosen in [2]. We note that we skip the VQA datasets in our retraining of JAT/Gato, and use a smaller batch size to fit available GPUs. For JAT/Gato (All Data) we train over the full JAT dataset for the specified 250k training steps. For JAT/Gato we train on the REGENT subset (which is 5-10x smaller) for 25k training steps. In both variants, the training loss converges to the same low value at about 5000 steps into the training run.

Finetuning hyperparemeters: Starting from a pre-trained checkpoint, we finetune using the same optimizer as pre-training time but starting with 1/10th the learning rate (*i.e.,* 5e-6) and for 3 epochs over the finetuning demonstrations. We use this same recipe across all environments and for both REGENT and JAT/Gato. We only use 3 epochs to prevent overwriting of any capabilities learned during pre-training.

Train-from-scratch baseline: For the policy in this baseline, we use the Impala CNN [43] in Atari environments and a MLP with two hidden layers with 256 neurons each in Metaworld and Mujoco environments. We use the same learning rate (5e-5) for the same 3 epochs with the same optimizer

(AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.999$) as used for REGENT except we use a constant learning rate schedule.

Designating retrieval demonstrations in each training environment's dataset: In the JAT/Gato setting, we designate 100 random demonstrations from the each of training vector observation environments as the retrieval subset for that environment. We also designate 5 random demonstrations from the training image environments as the retrieval subsets for said environments. In the ProcGen setting, with only 20 demonstrations per level per environment, we designate all 20 as the retrieval set for that level in that environment.

We use all states in all demonstrations in each training environment's dataset as query states. So, when a query state is from a demonstration in the designated retrieval subset, we simply retrieve from all other demonstrations in the designated retrieval subset except the current one. When a query state is from a demonstration not in the designated retrieval subset, we retrieve simply from the designated retrieval subset. This way, we ensure that none of the retrieved states are from the same demonstration as the query state. A possible direction for future work includes intelligently designating and collecting demonstrations for retrieval (perhaps using ideas like persistency of excitation [44], memory-based methods [45, 33, 46, 47], expert intervention methods [48]).

Choice of held-out environments: In the JAT/Gato setting, for the unseen environments, we hold-out the 5 Metaworld environments recommended in [49], the 5 Atari environments held-out in [4], and finally, we choose two Mujoco environments of mixed difficulty (see Figure 1). In the ProcGen setting, following MTT [3], we hold-out 5 environments.

In this work, we generalize to unseen environments from the same suites as the training environments. In future work, it would be interesting to examine what is needed for generalization to new suites (such as more manipulation suites [50], quadrotor, driving simulators [51, 52, 53], biological simulation [54], etc.).

B Proof of Theorem 5.2

To prove theorem 5.2, we first state and prove the following lemma.

Let us refer to the family of policies represented by $\pi^{\theta}_{\texttt{REGENT}}(a|s,r,c)$ in Equation (1) for various θ as $\Pi^{\theta}(a|s,r,c)$. The parameters θ of the transformer are drawn from the space Θ.

Lemma B.1. *(Total Variation in the Policy Class) The total variation of the policy class $\Pi^{\theta}(a|s,r,c)$ in environment j, $\forall\ \theta_1, \theta_2 \in \Theta$ and for some $s \in \mathcal{S}_j$, $r \in \mathcal{R}_j$, $c \in \mathcal{C}_j$, defined as $\sup_{\theta_1,\theta_2} \sup_{a} |\pi^{\theta_1}_{\texttt{REGENT}}(a|s,r,c) - \pi^{\theta_2}_{\texttt{REGENT}}(a|s,r,c)|$, is upper bounded by $(1 - e^{-\lambda d^{I}_{\mathcal{D}_j}})$.*

Proof: We have,

$$
\begin{aligned}
&\sup_{\theta_1,\theta_2} \sup_{a} |\pi^{\theta_1}_{\texttt{REGENT}}(a|s,r,c) - \pi^{\theta_2}_{\texttt{REGENT}}(a|s,r,c)| \\
&\qquad = \sup_{\theta_1,\theta_2} \sup_{a} (1 - e^{-\lambda d}) \Big(\mathsf{Softmax}(\pi_{\theta_1}(a|s,r,c)) - \mathsf{Softmax}(\pi_{\theta_2}(a|s,r,c)) \Big) && (4) \\
&\qquad = (1 - e^{-\lambda d}) \sup_{\theta_1,\theta_2} \sup_{a} \Big(\mathsf{Softmax}(\pi_{\theta_1}(a|s,r,c)) - \mathsf{Softmax}(\pi_{\theta_2}(a|s,r,c)) \Big) && (5) \\
&\qquad \leq (1 - e^{-\lambda d^{I}_{\mathcal{D}_j}}) && (6)
\end{aligned}
$$

The first equality holds because for the same tuple of (state, previous reward, context), the R&P component does not change since it is not affected by the choice of parameters. The last inequality can be interpreted as a property of Softmax and the distance to the most isolated state. □

Now, we rewrite theorem 5.2 below and prove it.

Theorem B.2. *The sub-optimality gap in environment j is* $\mathbf{J}(\pi^{*}_{\mathbf{j}}) - \mathbf{J}(\pi^{\theta}_{\texttt{REGENT}}) \leq \mathbf{min}\{\mathbf{H}, \mathbf{H^2}(\mathbf{1} - \mathbf{e}^{-\lambda \mathbf{d}^{\mathbf{I}}_{\mathcal{D}_{\mathbf{j}}}})\}$

Proof: Recall that in imitation learning, if the population total variation (TV) risk $\mathbb{T}(\hat{\pi}, \pi^*) \leq \epsilon$, then, $J(\pi^*) - J(\hat{\pi}) \leq min\{H, H^2\epsilon\}$ (See [55] Lemma 4.3). Using this with Lemma B.1 proves the theorem. $\square$

The main consequence of this theorem, also observed in our results, is that the sub-optimality gap reduces with more demonstrations in $\mathcal{D}_j$ because of the reduced distance to the most isolated state.

C Additional JAT/Gato Results

Generalization to unseen Mujoco environments/embodiments: In the two unseen Mujoco tasks shown in Figure 9, R&P appears to be a strong baseline even at generalizing to new embodiments! Moreover, after finetuning on just 25 demonstrations, only REGENT, not JAT/Gato, significantly improves to outperform other methods and only continues to get better with more demonstrations.

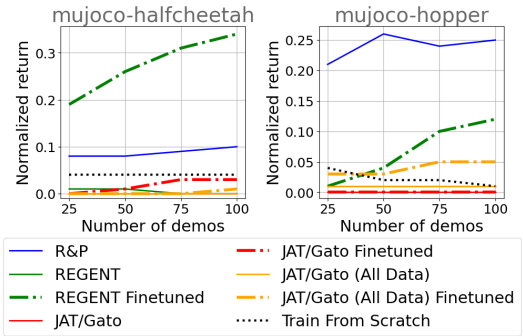

Figure 9: Normalized returns in the unseen Mujoco environments against the number of demonstration trajectories the agent can retrieve from or finetune on. Each value is an average across 100 rollouts of different seeds.

Additional qualitative examples: We plot examples of the inputs and outputs of REGENT at various states during a rendered rollout in the metaworld-bin-picking environment in Figure 10. In this continuous Metaworld example, REGENT can be seen predicting actions that are somewhat similar to but not the same as the first retrieved (R&P) action. These differences lead to better overall performance as seen in Figure 4. These differences are also the direct result of REGENT's in-context learning capabilities, learned from the preprocessed datasets from the pre-training environments.

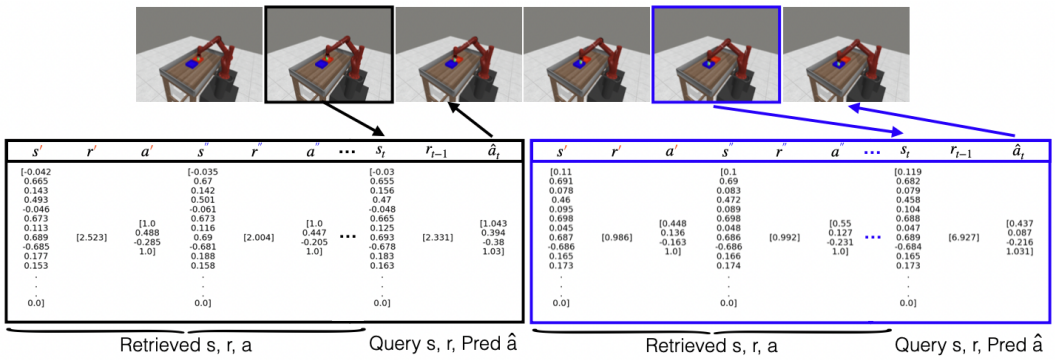

Figure 10: Qualitative examples of a few inputs and outputs of REGENT for various states in a rendered rollout in the unseen, continuous action space, metaworld-bin-picking environment.

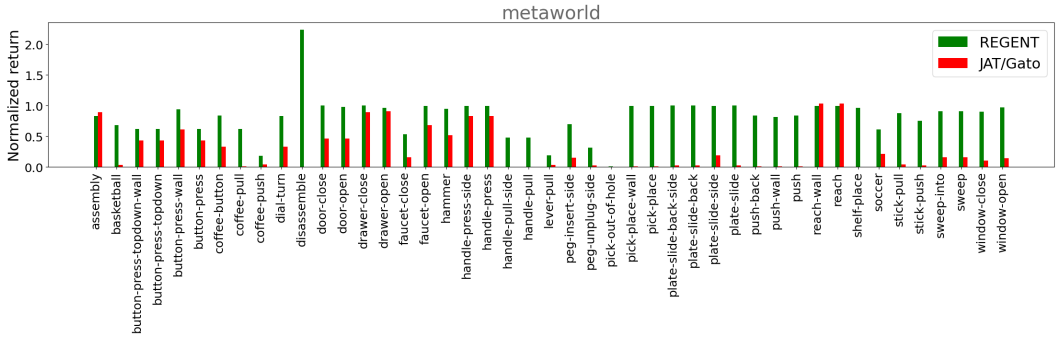

Figure 11: Normalized returns in each of the 45 seen Metaworld envs. Each value is an average across 100 rollouts of different seeds. See Table 3 for all values.

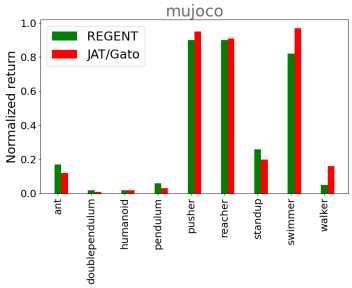

Figure 12: Normalized returns in each of the 9 seen Mujoco envs. Each value is an average across 100 rollouts of different seeds. See Table 4 for all values.

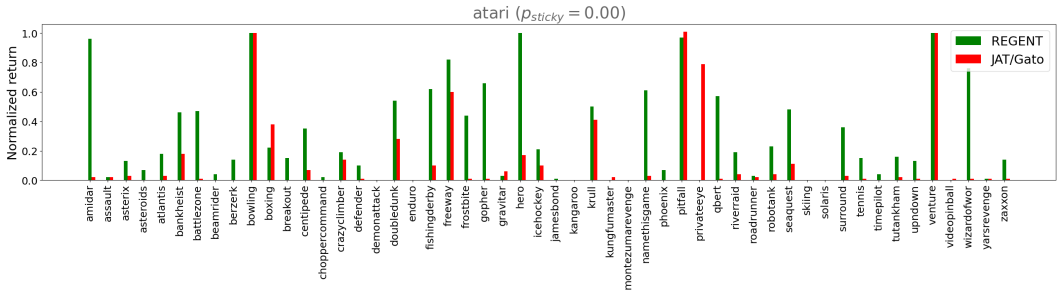

Figure 13: Normalized returns in each of the 52 seen Atari envs. Each value is an average across 15 rollouts of different seeds. See Table 5 for all values.

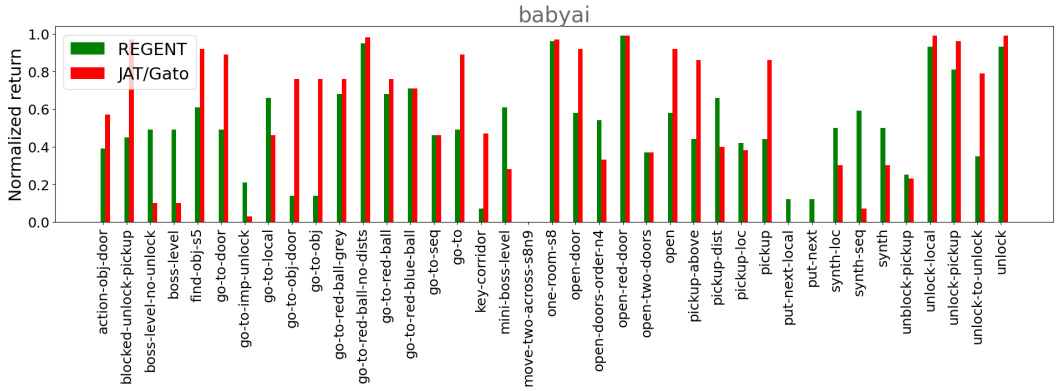

Figure 14: Normalized returns in each of the 45 seen BabyAI envs. Each value is an average across 100 rollouts of different seeds. See Table 6 for all values.

Env	Num demos	R&P	REGENT	REGENT Finetuned	JAT/Gato	JAT/Gato Finetuned	JAT/Gato (All Data)	JAT/Gato (All Data) Finetuned	Train From Scratch
metaworld-bin-picking	25	0.33	0.19	0.27	0.0	0.0	0.0	0.0	0.0
metaworld-bin-picking	50	0.4	0.53	0.6	0.0	0.0	0.0	0.0	0.0
metaworld-bin-picking	75	0.34	0.53	0.66	0.0	0.01	0.0	0.01	0.0
metaworld-bin-picking	100	0.33	0.65	0.73	0.0	0.0	0.0	0.01	0.0
metaworld-box-close	25	0.58	0.58	0.76	0.0	0.0	0.0	0.0	0.01
metaworld-box-close	50	0.69	0.69	0.86	0.0	0.0	0.0	0.0	0.0
metaworld-box-close	75	0.68	0.82	0.98	0.0	0.0	0.0	0.0	0.0
metaworld-box-close	100	0.81	0.86	0.97	0.0	0.0	0.0	0.0	0.0
metaworld-door-lock	25	0.58	0.72	0.8	0.13	0.12	0.24	0.31	0.0
metaworld-door-lock	50	0.55	0.78	0.83	0.13	0.15	0.24	0.25	0.03
metaworld-door-lock	75	0.63	0.83	0.83	0.13	0.17	0.24	0.27	0.01
metaworld-door-lock	100	0.64	0.83	0.85	0.13	0.17	0.24	0.28	0.03
metaworld-door-unlock	25	0.53	0.58	0.67	0.01	0.03	0.06	0.07	0.0
metaworld-door-unlock	50	0.55	0.75	0.82	0.01	0.03	0.06	0.07	0.01
metaworld-door-unlock	75	0.78	0.87	0.91	0.01	0.02	0.06	0.05	0.0
metaworld-door-unlock	100	0.86	0.89	0.93	0.01	0.03	0.06	0.08	0.03
metaworld-hand-insert	25	0.27	0.6	0.55	0.01	0.07	0.01	0.05	0.0
metaworld-hand-insert	50	0.54	0.66	0.68	0.01	0.04	0.01	0.06	0.0
metaworld-hand-insert	75	0.56	0.76	0.79	0.01	0.04	0.01	0.06	0.0
metaworld-hand-insert	100	0.69	0.76	0.83	0.01	0.02	0.01	0.03	0.01
atari-pong	7 (11599)	0.08	0.08	0.2	0.0	0.0	0.0	0.0	0.0
atari-pong	10 (16533)	0.18	0.2	0.23	0.0	0.0	0.0	0.0	0.0
atari-pong	13 (21468)	0.18	0.22	0.22	0.0	0.0	0.0	0.0	0.0
atari-pong	16 (26400)	0.18	0.22	0.25	0.0	0.0	0.0	0.0	0.0
atari-alien	3 (12759)	0.04	0.08	0.09	0.0	0.0	0.0	0.0	0.0
atari-alien	4 (17353)	0.05	0.08	0.08	0.0	0.0	0.0	0.0	0.0
atari-alien	5 (22684)	0.05	0.08	0.1	0.0	0.0	0.0	0.0	0.0
atari-alien	6 (27692)	0.05	0.09	0.1	0.0	0.0	0.0	0.0	0.0
atari-mspacman	4 (11902)	0.68	0.84	0.83	0.0	0.0	0.01	0.02	0.0
atari-mspacman	5 (14520)	0.68	0.83	0.83	0.0	0.01	0.01	0.02	0.0
atari-mspacman	9 (22490)	0.82	0.86	0.9	0.0	0.02	0.01	0.01	0.0
atari-mspacman	10 (25160)	0.82	0.86	0.9	0.0	0.01	0.01	0.02	0.0
Aggregate Mean		0.473	0.572	0.627	0.019	0.029	0.04	0.052	0.004

Table 1: Values in Figure 4. Each value is an average across 100 rollouts of different seeds in Metaworld and 15 rollouts of different seeds in Atari.

Env	Num demos	R&P	REGENT	REGENT Finetuned	JAT/Gato	JAT/Gato Finetuned	JAT/Gato (All Data)	JAT/Gato (All Data) Finetuned	Train From Scratch
mujoco-halfcheetah	25	0.08	0.01	0.19	0.0	0.0	0.0	0.0	0.04
mujoco-halfcheetah	50	0.08	0.01	0.26	0.0	0.01	0.0	0.0	0.04
mujoco-halfcheetah	75	0.09	0.0	0.31	0.0	0.03	0.0	0.0	0.04
mujoco-halfcheetah	100	0.1	0.0	0.34	0.0	0.03	0.0	0.01	0.04
mujoco-hopper	25	0.21	0.01	0.01	0.0	0.0	0.01	0.03	0.04
mujoco-hopper	50	0.26	0.01	0.04	0.0	0.0	0.01	0.03	0.02
mujoco-hopper	75	0.24	0.01	0.1	0.0	0.0	0.01	0.05	0.02
mujoco-hopper	100	0.25	0.01	0.12	0.0	0.0	0.01	0.05	0.01
Aggregate Mean		0.164	0.008	0.171	0.0	0.009	0.005	0.021	0.031

Table 2: Values in Figure 9. Each value is an average across 100 rollouts of different seeds.

Env	Num demos	REGENT	JAT/Gato
metaworld-assembly	100	0.83	0.89
metaworld-basketball	100	0.68	0.03
metaworld-button-press-topdown-wall	100	0.62	0.43
metaworld-button-press-topdown	100	0.62	0.43
metaworld-button-press-wall	100	0.94	0.61
metaworld-button-press	100	0.62	0.43
metaworld-coffee-button	100	0.84	0.33
metaworld-coffee-pull	100	0.62	0.01
metaworld-coffee-push	100	0.18	0.04
metaworld-dial-turn	100	0.83	0.33
metaworld-disassemble	100	2.24	0.0
metaworld-door-close	100	1.0	0.46
metaworld-door-open	100	0.98	0.46
metaworld-drawer-close	100	1.0	0.89
metaworld-drawer-open	100	0.96	0.91
metaworld-faucet-close	100	0.53	0.16
metaworld-faucet-open	100	0.99	0.68
metaworld-hammer	100	0.95	0.52
metaworld-handle-press-side	100	0.99	0.83
metaworld-handle-press	100	0.99	0.83
metaworld-handle-pull-side	100	0.48	0.0
metaworld-handle-pull	100	0.48	0.0
metaworld-lever-pull	100	0.19	0.03
metaworld-peg-insert-side	100	0.7	0.15
metaworld-peg-unplug-side	100	0.31	0.02
metaworld-pick-out-of-hole	100	0.01	0.0
metaworld-pick-place-wall	100	0.99	0.01
metaworld-pick-place	100	0.99	0.01
metaworld-plate-slide-back-side	100	1.0	0.02
metaworld-plate-slide-back	100	1.0	0.02
metaworld-plate-slide-side	100	0.99	0.19
metaworld-plate-slide	100	1.0	0.02
metaworld-push-back	100	0.84	0.01
metaworld-push-wall	100	0.81	0.01
metaworld-push	100	0.84	0.01
metaworld-reach-wall	100	0.99	1.03
metaworld-reach	100	0.99	1.03
metaworld-shelf-place	100	0.96	0.0
metaworld-soccer	100	0.61	0.21
metaworld-stick-pull	100	0.88	0.04
metaworld-stick-push	100	0.75	0.02
metaworld-sweep-into	100	0.91	0.16
metaworld-sweep	100	0.91	0.16
metaworld-window-close	100	0.9	0.1
metaworld-window-open	100	0.97	0.14
Aggregate Mean		0.82	0.281

Table 3: Values in Figure 11. Each value is an average across 100 rollouts of different seeds.

Env	Num demos	REGENT	JAT/Gato
mujoco-ant	100	0.17	0.12
mujoco-doublependulum	100	0.02	0.01
mujoco-humanoid	100	0.02	0.02
mujoco-pendulum	100	0.06	0.03
mujoco-pusher	100	0.9	0.95
mujoco-reacher	100	0.9	0.91
mujoco-standup	100	0.26	0.2
mujoco-swimmer	100	0.82	0.97
mujoco-walker	100	0.05	0.16
Aggregate Mean		0.356	0.374

Table 4: Values in Figure 12. Each value is an average across 100 rollouts of different seeds.

Env	Num demos	REGENT	JAT/Gato
atari-amidar	5	0.96	0.02
atari-assault	5	0.02	0.02
atari-asterix	5	0.13	0.03
atari-asteroids	5	0.07	0.0
atari-atlantis	5	0.18	0.03
atari-bankheist	5	0.46	0.18
atari-battlezone	5	0.47	0.01
atari-beamrider	5	0.04	0.0
atari-berzerk	5	0.14	0.0
atari-bowling	5	1.0	1.0
atari-boxing	5	0.22	0.38
atari-breakout	5	0.15	0.0
atari-centipede	5	0.35	0.07
atari-choppercommand	5	0.02	0.0
atari-crazyclimber	5	0.19	0.14
atari-defender	5	0.1	0.01
atari-demonattack	5	0.0	0.0
atari-doubledunk	5	0.54	0.28
atari-enduro	5	0.0	0.0
atari-fishingderby	5	0.62	0.1
atari-freeway	5	0.82	0.6
atari-frostbite	5	0.44	0.01
atari-gopher	5	0.66	0.01
atari-gravitar	5	0.03	0.06
atari-hero	5	1.0	0.17
atari-icehockey	5	0.21	0.1
atari-jamesbond	5	0.01	0.0
atari-kangaroo	5	0.0	0.0
atari-krull	5	0.5	0.41
atari-kungfumaster	5	0.0	0.02
atari-montezumarevenge	5	0.0	0.0
atari-namethisgame	5	0.61	0.03
atari-phoenix	5	0.07	0.0
atari-pitfall	5	0.97	1.01
atari-privateeye	5	0.0	0.79
atari-qbert	5	0.57	0.01
atari-riverraid	5	0.19	0.04
atari-roadrunner	5	0.03	0.02
atari-robotank	5	0.23	0.04
atari-seaquest	5	0.48	0.11
atari-skiing	5	0.0	0.0
atari-solaris	5	0.0	0.0
atari-surround	5	0.36	0.03
atari-tennis	5	0.15	0.01
atari-timepilot	5	0.04	0.0
atari-tutankham	5	0.16	0.02
atari-upndown	5	0.13	0.01
atari-venture	5	1.0	1.0
atari-videopinball	5	0.0	0.01
atari-wizardofwor	5	0.76	0.01
atari-yarsrevenge	5	0.01	0.01
atari-zaxxon	5	0.14	0.01
Aggregate Mean		0.293	0.131

Table 5: Values in Figure 13. Each value is an average across 15 rollouts of different seeds.

Env	Num demos	REGENT	JAT/Gato
babyai-action-obj-door	20	0.39	0.57
babyai-blocked-unlock-pickup	20	0.45	0.97
babyai-boss-level-no-unlock	20	0.49	0.1
babyai-boss-level	20	0.49	0.1
babyai-find-obj-s5	20	0.61	0.92
babyai-go-to-door	20	0.49	0.89
babyai-go-to-imp-unlock	20	0.21	0.03
babyai-go-to-local	20	0.66	0.46
babyai-go-to-obj-door	20	0.14	0.76
babyai-go-to-obj	20	0.14	0.76
babyai-go-to-red-ball-grey	20	0.68	0.76
babyai-go-to-red-ball-no-dists	20	0.95	0.98
babyai-go-to-red-ball	20	0.68	0.76
babyai-go-to-red-blue-ball	20	0.71	0.71
babyai-go-to-seq	20	0.46	0.46
babyai-go-to	20	0.49	0.89
babyai-key-corridor	20	0.07	0.47
babyai-mini-boss-level	20	0.61	0.28
babyai-move-two-across-s8n9	20	0.0	0.0
babyai-one-room-s8	20	0.96	0.97
babyai-open-door	20	0.58	0.92
babyai-open-doors-order-n4	20	0.54	0.33
babyai-open-red-door	20	0.99	0.99
babyai-open-two-doors	20	0.37	0.37
babyai-open	20	0.58	0.92
babyai-pickup-above	20	0.44	0.86
babyai-pickup-dist	20	0.66	0.4
babyai-pickup-loc	20	0.42	0.38
babyai-pickup	20	0.44	0.86
babyai-put-next-local	20	0.12	0.0
babyai-put-next	20	0.12	0.0
babyai-synth-loc	20	0.5	0.3
babyai-synth-seq	20	0.59	0.07
babyai-synth	20	0.5	0.3
babyai-unblock-pickup	20	0.25	0.23
babyai-unlock-local	20	0.93	0.99
babyai-unlock-pickup	20	0.81	0.96
babyai-unlock-to-unlock	20	0.35	0.79
babyai-unlock	20	0.93	0.99
Aggregate Mean		0.508	0.577

Table 6: Values in Figure 14. Each value is an average across 100 rollouts of different seeds.

D Additional ProcGen Results

Discussion about MTT's performance: We note that it is likely that MTT performance is close to random on a couple of environments only because the authors are forced to constrain MTT rollouts to 200 steps, stopping short of the true horizon of these environments, simply because of the quadratically increasing complexity of attention with increase in context size. MTT also likely does not report results beyond 4 demonstrations in the context for this reason. Yet, this is not an issue for `R&P` or `REGENT` which can scale to any number of demonstrations to retrieve from since it only uses the 19 closest states in the context.

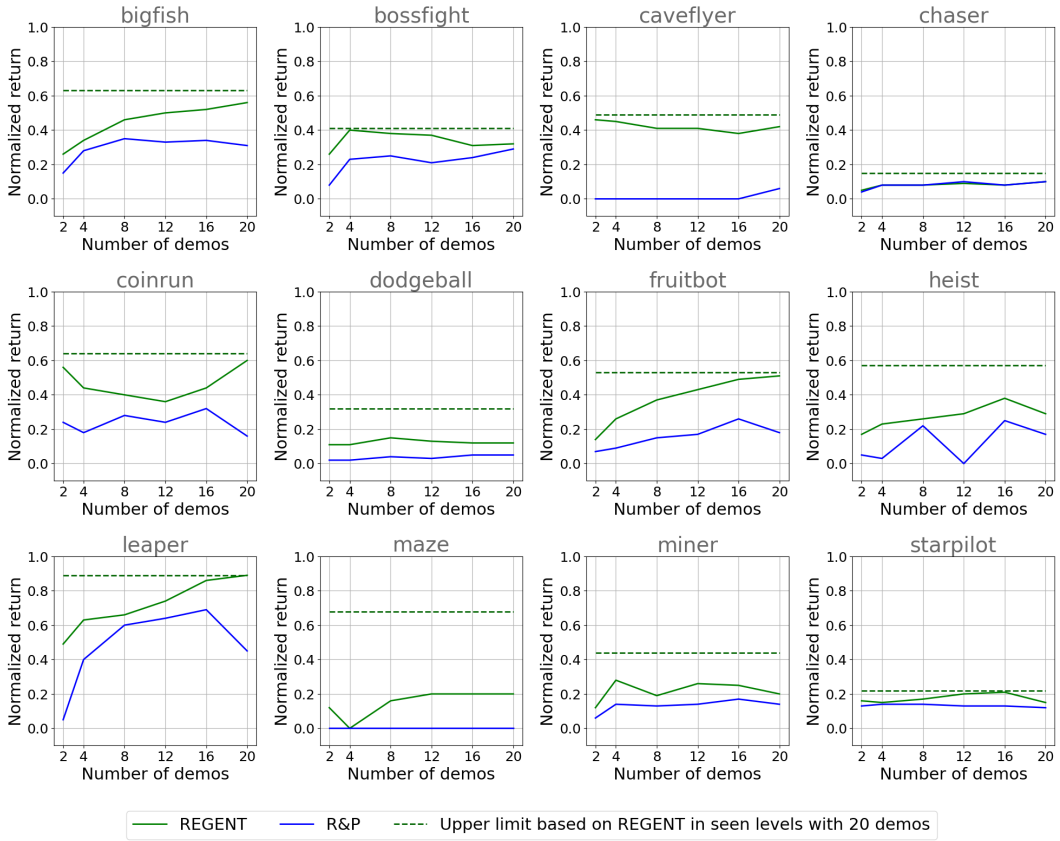

Figure 15: Normalized returns in unseen levels in all 12 ProcGen training environments against the number of demonstration trajectories the agent can retrieve from. Each value is an average across 10 levels with 5 rollouts each with $p_{\text{sticky}} = 0.1$. See Table 8 for all values.

Env	sticky prob	num demos	(epoch, end batch)	REGENT	R&P	MTT [3]
ninja	p=0.2	n=2	(0, 11612)	(7.2, 1.17)	(5.67, 1.1)	
		n=4	(0, 11612)	(7.4, 2.06)	(5.5, 1.25)	(4.1, 0.35)
		n=8	(0, 11612)	(7.4, 1.02)	(6.4, 1.5)	
		n=12	(0, 11612)	(7.8, 0.4)	(6.33, 1.2)	
		n=16	(0, 11612)	(8.0, 0.63)	(7.4, 1.11)	
		n=20	(0, 11612)	(8.2, 0.75)	(7.6, 1.2)	
climber	p=0.2	n=2	(0, 11612)	(4.98, 0.85)	(3.17, 0.93)	
		n=4	(0, 11612)	(6.52, 2.06)	(3.95, 1.42)	(1.85, 0.41)
		n=8	(0, 11612)	(6.38, 1.52)	(3.12, 1.87)	
		n=12	(0, 11612)	(6.92, 1.19)	(5.64, 2.37)	
		n=16	(0, 11612)	(6.66, 1.18)	(3.77, 2.0)	
		n=20	(0, 11612)	(7.04, 1.65)	(5.41, 1.78)	
plunder	p=0.2	n=2	(0, 11612)	(10.98, 2.23)	(8.46, 1.26)	
		n=4	(0, 11612)	(11.9, 0.77)	(8.3, 1.88)	(2.5, 0.2)
		n=8	(0, 11612)	(12.14, 1.22)	(8.2, 1.72)	
		n=12	(0, 11612)	(13.8, 1.59)	(9.36, 2.16)	
		n=16	(0, 11612)	(14.04, 2.63)	(9.84, 2.11)	
		n=20	(0, 11612)	(13.78, 1.81)	(10.8, 1.87)	
jumper	p=0.2	n=2	(0, 11612)	(4.6, 0.8)	(3.75, 1.45)	
		n=4	(0, 11612)	(5.8, 1.47)	(4.0, 1.28)	(4.65, 0.28)
		n=8	(0, 11612)	(6.0, 1.26)	(4.25, 1.69)	
		n=12	(0, 11612)	(6.4, 0.49)	(5.25, 0.92)	
		n=16	(0, 11612)	(6.8, 0.75)	(6.0, 1.02)	
		n=20	(0, 11612)	(6.8, 1.6)	(5.5, 1.54)	

Table 7: Values in Figure 5 before normalization. The tuples represent (mean, std) obtained across 10 levels with 5 rollouts each.

Env	sticky prob	num demos	(epoch, end batch)	REGENT	R&P	REGENT (seen levels)
bigfish	p=0.1	n=2	(0, 11612)	(11.32, 2.2)	(6.75, 3.92)	
		n=4	(0, 11612)	(14.32, 2.55)	(11.85, 3.09)	
		n=8	(0, 11612)	(18.82, 3.45)	(14.7, 4.08)	
		n=12	(0, 11612)	(20.66, 3.33)	(14.02, 4.4)	
		n=16	(0, 11612)	(21.32, 5.07)	(14.43, 3.24)	
		n=20	(0, 11612)	(22.74, 4.23)	(13.28, 3.28)	(25.7, 3.03)
bossfight	p=0.1	n=2	(0, 11612)	(3.76, 0.77)	(1.45, 1.13)	
		n=4	(0, 11612)	(5.52, 0.56)	(3.38, 1.4)	
		n=8	(0, 11612)	(5.24, 1.47)	(3.67, 1.1)	
		n=12	(0, 11612)	(5.12, 0.74)	(3.17, 1.4)	
		n=16	(0, 11612)	(4.32, 1.98)	(3.51, 1.13)	
		n=20	(0, 11612)	(4.44, 1.23)	(4.1, 1.38)	(5.6, 1.12)
caveflyer	p=0.1	n=2	(0, 11612)	(7.44, 1.13)	(2.74, 0.92)	
		n=4	(0, 11612)	(7.34, 0.71)	(2.98, 1.13)	
		n=8	(0, 11612)	(7.0, 0.89)	(3.29, 0.69)	
		n=12	(0, 11612)	(6.96, 0.4)	(3.27, 1.27)	
		n=16	(0, 11612)	(6.72, 0.83)	(3.31, 1.1)	
		n=20	(0, 11612)	(7.08, 0.81)	(4.04, 0.85)	(7.68, 0.48)
chaser	p=0.1	n=2	(0, 11612)	(1.15, 0.2)	(1.01, 0.19)	
		n=4	(0, 11612)	(1.51, 0.43)	(1.51, 0.45)	
		n=8	(0, 11612)	(1.46, 0.16)	(1.47, 0.34)	
		n=12	(0, 11612)	(1.61, 0.11)	(1.77, 0.74)	
		n=16	(0, 11612)	(1.47, 0.08)	(1.53, 0.47)	
		n=20	(0, 11612)	(1.79, 0.47)	(1.73, 0.47)	(2.4, 0.47)
coinrun	p=0.1	n=2	(0, 11612)	(7.8, 1.6)	(6.2, 1.17)	
		n=4	(0, 11612)	(7.2, 0.75)	(5.9, 1.37)	
		n=8	(0, 11612)	(7.0, 0.89)	(6.4, 1.28)	
		n=12	(0, 11612)	(6.8, 0.98)	(6.2, 1.08)	
		n=16	(0, 11612)	(7.2, 1.17)	(6.6, 1.28)	
		n=20	(0, 11612)	(8.0, 1.1)	(5.8, 0.87)	(8.2, 0.4)
dodgeball	p=0.1	n=2	(0, 11612)	(3.36, 1.04)	(1.93, 0.93)	
		n=4	(0, 11612)	(3.44, 0.32)	(1.8, 0.86)	
		n=8	(0, 11612)	(4.04, 1.09)	(2.17, 0.54)	
		n=12	(0, 11612)	(3.72, 0.69)	(2.07, 0.72)	
		n=16	(0, 11612)	(3.64, 0.82)	(2.33, 0.67)	
		n=20	(0, 11612)	(3.56, 0.74)	(2.4, 0.6)	(7.12, 1.12)
fruitbot	p=0.1	n=2	(0, 11612)	(3.18, 0.7)	(0.98, 1.48)	
		n=4	(0, 11612)	(7.48, 2.44)	(1.6, 3.05)	
		n=8	(0, 11612)	(11.02, 1.94)	(3.67, 3.37)	
		n=12	(0, 11612)	(13.02, 1.39)	(4.38, 4.59)	
		n=16	(0, 11612)	(15.04, 2.26)	(7.45, 5.22)	
		n=20	(0, 11612)	(15.62, 1.76)	(4.67, 4.56)	(16.36, 3.49)
heist	p=0.1	n=2	(0, 11612)	(4.6, 0.8)	(3.8, 1.72)	
		n=4	(0, 11612)	(5.0, 2.83)	(3.7, 1.9)	
		n=8	(0, 11612)	(5.2, 1.72)	(4.9, 1.3)	
		n=12	(0, 11612)	(5.4, 1.2)	(3.4, 1.11)	
		n=16	(0, 11612)	(6.0, 0.63)	(5.1, 1.3)	
		n=20	(0, 11612)	(5.4, 1.36)	(4.6, 1.2)	(7.2, 1.17)
leaper	p=0.1	n=2	(0, 11612)	(6.4, 1.2)	(3.33, 1.73)	
		n=4	(0, 11612)	(7.4, 1.62)	(5.83, 1.35)	
		n=8	(0, 11612)	(7.6, 1.36)	(7.17, 1.0)	
		n=12	(0, 11612)	(8.2, 0.75)	(7.5, 1.43)	
		n=16	(0, 11612)	(9.0, 1.1)	(7.83, 1.0)	
		n=20	(0, 11612)	(9.2, 0.75)	(6.17, 0.78)	(9.2, 0.75)
maze	p=0.1	n=2	(0, 11612)	(5.6, 0.49)	(5.0, 1.41)	
		n=4	(0, 11612)	(4.8, 1.17)	(3.1, 0.94)	
		n=8	(0, 11612)	(5.8, 1.17)	(4.9, 0.83)	
		n=12	(0, 11612)	(6.0, 0.63)	(4.6, 0.8)	
		n=16	(0, 11612)	(6.0, 1.67)	(4.5, 1.02)	
		n=20	(0, 11612)	(6.0, 0.89)	(4.5, 1.12)	(8.4, 0.8)
miner	p=0.1	n=2	(0, 11612)	(2.88, 1.22)	(2.14, 0.72)	
		n=4	(0, 11612)	(4.72, 0.82)	(3.12, 0.54)	
		n=8	(0, 11612)	(3.68, 1.53)	(2.96, 1.46)	
		n=12	(0, 11612)	(4.54, 1.54)	(3.09, 1.05)	
		n=16	(0, 11612)	(4.38, 0.93)	(3.45, 1.04)	
		n=20	(0, 11612)	(3.76, 0.38)	(3.12, 1.11)	(6.58, 1.56)
starpilot	p=0.1	n=2	(0, 11612)	(12.52, 1.16)	(10.28, 2.07)	
		n=4	(0, 11612)	(11.94, 2.27)	(11.33, 2.16)	
		n=8	(0, 11612)	(12.82, 2.75)	(10.93, 2.19)	
		n=12	(0, 11612)	(14.9, 4.75)	(10.8, 1.03)	
		n=16	(0, 11612)	(15.34, 2.32)	(10.49, 1.85)	
		n=20	(0, 11612)	(11.8, 1.82)	(10.08, 2.25)	(16.1, 3.8)

Table 8: Values in Figures 6 and 15 before normalization. The tuples represent (mean, std) obtained across 10 levels with 5 rollouts each.