RETRIEVAL-AUGMENTED DECISION TRANSFORMER: EXTERNAL MEMORY FOR IN-CONTEXT RL

Anonymous authors
Paper under double-blind review

ABSTRACT

In-context learning (ICL) is the ability of a model to learn a new task by observing a few exemplars in its context. While prevalent in NLP, this capability has recently also been observed in Reinforcement Learning (RL) settings. Prior in-context RL methods, however, require entire episodes in the agent's context. Given that complex environments typically lead to long episodes with sparse rewards, these methods are constrained to simple environments with short episodes. To address these challenges, we introduce Retrieval-Augmented Decision Transformer (RA-DT). RA-DT employs an external memory mechanism to store past experiences from which it retrieves only sub-trajectories relevant for the current situation. The retrieval component in RA-DT does not require training and can be entirely domain-agnostic. We evaluate the capabilities of RA-DT on grid-world environments, robotics simulations, and procedurally-generated video games. On grid-worlds, RA-DT outperforms baselines, while using only a fraction of their context length. Furthermore, we illuminate the limitations of current in-context RL methods on complex environments and discuss future directions. To facilitate future research, we release datasets for four of the considered environments.

1 INTRODUCTION

In-context Learning (ICL) is the ability of a model to learn new tasks by leveraging a few exemplars in its context [Brown et al., 2020]. Large Language Models (LLMs) exhibit this capability after pre-training on large amounts of data crawled from the web. A similar trend has emerged in the field of RL, where agents are pre-trained on datasets with an increasing number of tasks [Chen et al., 2021; Janner et al., 2021; Reed et al., 2022; Lee et al., 2022; Brohan et al., 2022; 2023]. After training, such an agent is capable of learning new tasks by observing previous trials in its context [Laskin et al., 2022; Liu & Abbeel, 2023; Lee et al., 2023; Raparthy et al., 2023]. Consequently, ICL is a promising direction for generalist agents to acquire new tasks without the need for re-training, fine-tuning, or providing expert-demonstrations.

Existing methods for in-context RL rely on keeping entire episodes in their context [Laskin et al., 2022; Lee et al., 2023; Kirsch et al., 2023; Raparthy et al., 2023]. Consequently, these methods face challenges in complex environments, as complex environments are usually characterized by long episodes and sparse rewards. Episodes in RL may consist of thousands of interaction steps, and processing them is computationally expensive, especially for network architectures such as the Transformer [Vaswani et al., 2017]. Furthermore, not all information an agent encountered in the past may be necessary to solve the new task. Therefore, we address the question of how to facilitate ICL for environments with long episodes and sparse rewards.

We introduce Retrieval-Augmented Decision Transformer (RA-DT), which incorporates an external memory into the Decision Transformer [Chen et al., 2021, DT] architecture (see Figure 1). Our external memory enables efficient storage and retrieval of past experiences, that are relevant for the current situation. We achieve this by leveraging a vector index populated with sub-trajectories, in combination with maximum inner product search; akin to Retrieval-augmented Generation (RAG) in LLMs [Khandelwal et al., 2019; Lewis et al., 2020; Borgeaud et al., 2022]. To encode retrieved sub-trajectories, RA-DT relies on a pre-trained embedding model, which can either be domain-specific, such as a DT trained on the same domain, or a domain-agnostic language model (LM) (see Section 3). Subsequently, RA-DT uses cross-attention to leverage the retrieved sub-trajectories and predict

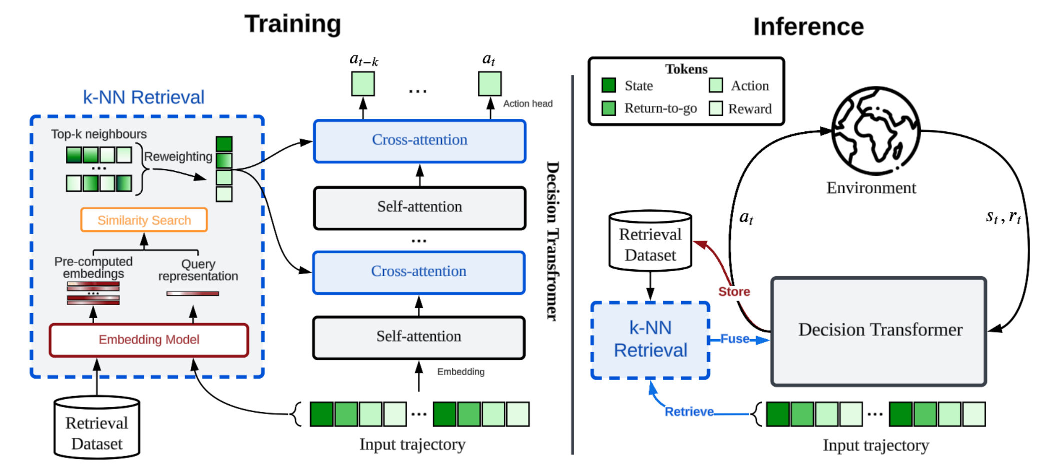


Figure 1: Illustration of **Retrieval-augmented Decision Transformer (RA-DT)**. **Left:** Prior to training, we encode pre-collected trajectories via an embedding model. During training, we retrieve sub-trajectories using the current context as query, and fuse them into layers via cross-attention. **Right:** During inference, the collected experience is stored and retrieved during environment interaction.

the next action. This way, RA-DT does not rely on a long context and can deal with sparse reward settings.

We evaluate the effectiveness of RA-DT on grid-world environments used in prior work with sparse rewards and increasing grid-sizes (Dark-Room, Dark Key-Door, Maze-Runner), robotics environments (Meta-World, DMControl) and procedurally-generated video games (Procgen). On grid-worlds, RA-DT considerably outperforms previous in-context RL methods, while only using a fraction of their context length. Further, we show that our domain-agnostic trajectory embedding model reaches performance close to a domain-specific one. On the remaining more complex environments, we observe consistent improvements for RA-DT on hold-out tasks, but no in-context improvement for any method. Therefore, we discuss the current limitations of RA-DT and other in-context RL methods and elaborate on potential remedies and future directions for in-context RL.

We make the following **contributions**:

- We introduce Retrieval-augmented Decision Transformers (RA-DT) and evaluate its effectiveness on a number of diverse domains.
- We show that a domain-agnostic embedding model can be utilized for retrieval in RL without requiring any pre-training, and reaches performance close to a domain-specific model.
- We release datasets for Dark-Room, Dark Key-Door, Maze-Runner, and Procgen to foster future research on in-context decision-making that leverages offline pre-training.

2 Related Work

In-context Learning. ICL is a form of Meta-learning, also referred to as learning-to-learn [Schmidhuber, 1987]. Typically, meta-learning is *targeted* and learned through a meta-training phase, for example in supervised-learning [Santoro et al., 2016; Mishra et al., 2018; Finn et al., 2017] or in RL [Wang et al., 2016; Duan et al., 2016; Kirsch et al., 2019; Flennerhag et al., 2019]. In contrast, ICL *emerges* as a result of pre-training on a certain data distribution [Chan et al., 2022]. This ability was first observed in Hochreiter et al. [2001] via LSTMs [Hochreiter & Schmidhuber, 1997] and later re-discovered in LLMs [Brown et al., 2020]. Ortega et al. [2019] found that every memory-based architecture may exhibit such capabilities. Another crucial factor is a training distribution comprising a vast amount of tasks [Chan et al., 2022; Kirsch et al., 2022]. Recent works combined these properties to induce ICL in RL [Laskin et al., 2022; Lee et al., 2022; Kirsch et al., 2023]. While promising, they require keeping entire episodes in context, which is difficult in environments with long episodes. Raparthy et al. [2023] consider an in-context imitation learning setting given expert demonstrations. In contrast, RA-DT can handle long episodes and does not rely on expert demonstrations.

Retrieval-augmented Generation. The aim of retrieval-augmentation is to provide a model access to an external memory. This alleviates the need to store the training data in the parameters of a model and allows to condition on new data without re-training. RAG is successfully applied in the realm of LLMs [Khandelwal et al., 2019; Guu et al., 2020; Lewis et al., 2020; Borgeaud et al., 2022; Izacard et al., 2022; Ram et al., 2023], multi-modal language generation [Hu et al., 2023; Yasunaga et al., 2023; Yang et al., 2023b; Ramos et al., 2022], and for chemical reaction prediction [Seidl et al., 2022]. In RL, the access to an external memory is often referred to as episodic memory [Sprechmann et al., 2018; Blundell et al., 2016; Pritzel et al., 2017]. Goyal et al. [2022] investigate the effect of different data sources in the external memory of an online RL agent. [Humphreys et al., 2022] provide access to millions of expert demonstrations via RAG in the game of Go. In contrast, RA-DT does not rely on expert demonstrations, but leverages RAG to learn new tasks entirely in-context without the need for weight updates. Further, RA-DT does *not* rely on a pre-trained domain-specific embedding model, as we demonstrate that the embedding model can be entirely domain-agnostic.

External memory in RL. Most prior works have explored the utility of an external memory to cope with partially observable environments [Åström, 1965; Kaelbling et al., 1998], in which the agent must remember past events to approximate the true state of the environment. This is difficult, especially for complex tasks with sparse rewards [Arjona-Medina et al., 2019; Patil et al., 2022; Widrich et al., 2021] and long episodes. To cope with this problem, Neural Turing Machines [Graves et al., 2014], which rely on a neural controller to read from and write to an external memory, were applied to RL [Zaremba & Sutskever, 2015]. Memory networks [Weston et al., 2015] leverage an external memory for reasoning. Wayne et al. [2018] propose a memory architecture with read/write access to learn what information to store based on a world model. In contrast, RA-DT only retrieves pieces of past information similar to the current encountered situation. Hill et al. [2021] propose an attention-based external memory, where queries, keys, and values are represented by different modalities. Similarly, our domain-agnostic embedding model extends the idea of history compression via LLMs [Paischer et al., 2022; 2023] to retrieval, where queries and keys are encoded in the language space, while values comprise raw sub-trajectories.

3 Method

3.1 Background

Reinforcement Learning. We formulate our problem setting as a Markov Decision Process (MDP) that is represented by a 4-tuple of$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. $\mathcal{S}$ and $\mathcal{A}$ denote state and action spaces, respectively. At timestep t the agent observes state $s_t \in \mathcal{S}$ and issues action $a_t \in \mathcal{A}$. For each executed action, the agent receives a scalar reward r_t, which is given by the reward function $\mathcal{R}(r_t \mid s_t, a_t)$. $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ constitutes a probability distribution over next states s_{t+1} when issuing action a_t in state s_t. RL aims at learning a policy $\pi(a_t \mid s_t)$ that predicts action a_t in state s_t that maximizes r_t.

Decision Transformer. Decision Transformer [Chen et al., 2021, DT] learns a policy from offline data by conditioning on future rewards. This allows rephrasing RL as a sequence modelling problem, where the agent is trained in a supervised manner to map future rewards to actions, often referred to as upside-down RL [Schmidhuber, 2019]. To train the DT, we assume access to a pre-collected dataset $\mathcal{D} = \{\tau_i \mid 1 \leq i \leq N\}$ of N trajectories τ_i that are sampled from the environment via a behavioural policy π_β. Each trajectory $\tau \in \mathcal{D}$ consists of state, action, reward, and return-to-go (RTG) quadruplets $\tau_i = (s_0, a_0, r_0, \hat{R}_0, \dots, s_T, a_T, r_T, \hat{R}_T)$, where T represents the length of trajectory τ_i, and $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. The DT π_θ is trained to predict the ground truth action a_t conditioned on sub-trajectories via cross-entropy or mean-squared error loss, depending on the domain:

$$a_t \sim \pi_\theta(a_t \mid s_{t-C:t}, \hat{R}_{t-C:t}, a_{t-C:t-1}, r_{t-C:t-1}), \tag{1}$$

where $C \leq T$ is the context length. During inference, the DT is conditioned on a high RTG to produce a likely sequence of actions that yields high reward behaviour.

3.2 Retrieval-augmented Decision Transformer (RA-DT)

Processing long sequences with DTs is computationally expensive due to the quadratic complexity of the Transformer architecture. To address this challenge, we introduce RA-DT, which equips the DT

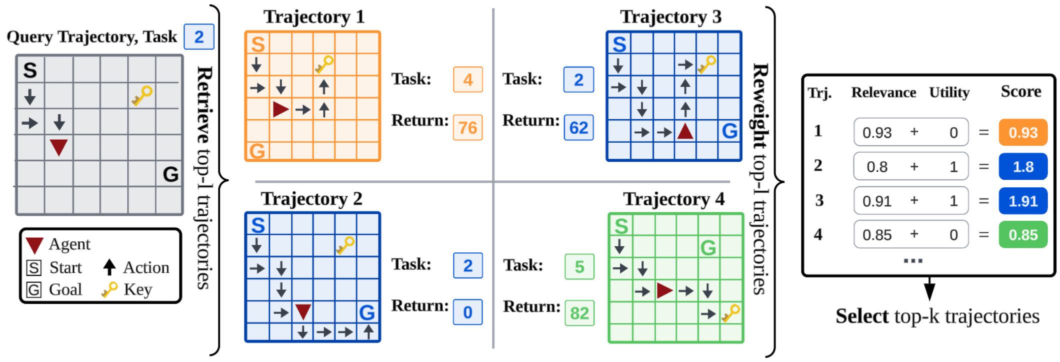


Figure 2: Illustration of **experience reweighting**. Given a query trajectory, we retrieve the top $l > k$ most *relevant* experiences by maximum inner product search. Each experience has an associated task ID, and return, based on which we compute their *utility*. We reweight by s_{rel} and s_u, to obtain the final retrieval score s_{ret}, and return the top-k experiences.

with an external memory that relies on a vector index for retrieval. Consequently, RA-DT consists of a parametric and a non-parametric component, reminiscent of complementary learning systems [Mcclelland et al., 1995; Kumaran et al., 2016]. The former is represented by the DT and learns to predict actions conditioned on the future return. The latter is the retrieval component that searches for relevant experiences, similar to Borgeaud et al. [2022] (see Figure 1).

3.2.1 Vector Index for Retrieval Augmentation

We aim at augmenting the DT with a vector index (external memory) that allows for retrieval of relevant experiences. To this end, we build our vector index by leveraging an embedding model $g : \tau \mapsto \mathbb{R}^{d_r}$ that takes a trajectory τ and returns a vector of size d_r. Given a dataset $\mathcal{D}$ of trajectories, we obtain a set of key-value pairs of our vector index by embedding all sub-trajectories $\tau_{t-C:t} \in \mathcal{D}$ via $g(\cdot)$ to obtain $\mathcal{K} \times \mathcal{V} = \{(g(\tau_{i,t-C:t}), \tau_{i,t-C:t+C}) \mid 1 \leq i \leq |\mathcal{D}|\}$. Note that values contain sub-trajectories ranging from $t - C$ to $t + C$, while keys use sub-trajectories $t - C : t$ for a fixed C, where t goes over trajectory length in increments of C (see Appendix C.4 for more details). The reason for this choice is that during inference, the model does not have access to future states.

In RAG applications for Natural Language Processing (NLP), a common choice for $g(\cdot)$ is a pre-trained LM. While pre-trained models in NLP are ubiquitous, they are rarely available in RL. A natural choice to instantiate $g(\cdot)$ is to train a DT on the pre-collected dataset $\mathcal{D}$, as they exhibit a well-separated embedding space after pre-training [Schmied et al., 2024]. Therefore, they are well suited for retrieval since a new task can be matched to similar tasks in the vector index. As a domain-agnostic alternative, we propose to utilize the FrozenHopfield (FH) mechanism Paischer et al. [2022] to map trajectories to the embedding space of a pre-trained LM. This enables instantiating $g(\cdot)$ with a pre-trained language encoder. The FH mechanism is parameterized by an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{v \times d_{\text{LM}}}$ of a pretrained LM with vocabulary size v and hidden dimension d_{LM}, a random matrix $\boldsymbol{P}$ with entries sampled from $\mathcal{N}(0, d_{\text{in}}/d_{\text{LM}})$, and a scaling factor β and performs:

$$\text{FH}(\boldsymbol{x}_t) = \boldsymbol{E}^\top \text{softmax}(\beta \boldsymbol{E}\boldsymbol{P}\boldsymbol{x}_t). \tag{2}$$

We denote $\boldsymbol{x}_t \in \mathbb{R}^{d_{\text{in}}}$ as the input token and apply the FH position-wise to every state/action/reward token in a sub-trajectory $\tau_{t-C:t}$ separately. Finally, we apply a LM on top of the FH to obtain the keys of our vector index by setting $g(\cdot) = \text{LM}(\text{FH}(\cdot))$. Utilizing the FH enables leveraging the expressive power of pre-trained LMs as trajectory encoders for RL. This sidesteps the need for pre-training a domain-specific model and can be incorporated in any existing retrieval-augmentation pipeline.

3.2.2 Searching for Similar Experiences

Given an input sub-trajectory $\tau_{\text{in}} \in \mathcal{D}$, we first construct a query $\boldsymbol{q} = g(\tau_{\text{in}})$, using our embedding model $g(\cdot)$ (see Appendix C.4 for details). Then, we use maximum inner product search (MIPS) between $\boldsymbol{q}$ and all keys $\boldsymbol{k} \in \mathcal{K}$ and select the corresponding top-l sub-trajectories $\tau_{\text{ret}} \in \mathcal{V}$ by:

$$\mathcal{R} = \arg\max_{\boldsymbol{k}\in\mathcal{K}}^{l} \operatorname{cossim}(\boldsymbol{q}, \boldsymbol{k}), \tag{3}$$

where $\operatorname{cossim}(\boldsymbol{q}, \boldsymbol{k}) = \frac{\mathbf{q}\cdot\mathbf{k}}{\|\mathbf{q}\|\|\mathbf{k}\|}$ is the cosine similarity. Consequently, $\mathcal{R}$ contains the set of retrieved sub-trajectories and their keys. Providing too similar experiences to the model may hinder learning [Yasunaga et al., 2023] and we apply retrieval regularization during training (see Appendix C.4).

3.2.3 Reweighting Retrieved Experiences

Following Park et al. [2023], we characterize the usefulness of retrieved sub-trajectories in $\mathcal{R}$ along two dimensions: *relevance* and *utility*. The relevance of a key $\boldsymbol{k} \in \mathcal{K}$ is defined by its cosine similarity to the query $\boldsymbol{q}$. While a retrieved experience may be relevant, it might not be important. Determining the utility of a sequence in general is hard. Thus, we experiment with two heuristics that follow different definitions of utility. The first assigns more utility to sub-trajectories with high return, and is utilized *at inference* only. The second assigns utility to sub-trajectories that originate from the same task as the query and is used *at training* only. Then, we reweight a retrieved experience according to:

$$s_{\text{ret}}(\boldsymbol{k}, \boldsymbol{q}, \tau_{\text{ret}}) = s_{\text{rel}}(\boldsymbol{k}, \boldsymbol{q}) + \alpha\, s_{\text{u}}(\tau_{\text{ret}}, \tau_{\text{in}}), \tag{4}$$

where $s_{rel} = \operatorname{cossim}(\boldsymbol{k}, \boldsymbol{q})$ and s_{u} measures the utility of a retrieved sub-trajectory weighted by α. Note that we instantiate $s_{\text{u}}(\cdot, \cdot)$ differently depending on whether the agent is in training or inference mode. At *training* time, a pre-collected set of trajectories that contains multiple tasks is stored in the vector index (Figure 1, left). Trajectories can be obtained from human demonstrations or RL agents. Therefore, we encourage the agent to retrieve sub-trajectories of the same task. During training, we use: $s_{\text{u}}(\tau_{\text{ret}}, \tau_{\text{in}}) = \mathbb{1}(\mathrm{t}(\tau_{\text{ret}}) = \mathrm{t}(\tau_{\text{in}}))$, where $\mathrm{t}(\cdot)$ takes a sub-trajectory and returns its task index.

During *inference*, we evaluate the ICL capabilities of the agent. Starting from an *empty* vector index, we store experiences of the agent while it interacts with the environment (see Figure 1, right). Thus, during inference, the agent can only retrieve experiences from the same task. Therefore, we steer the agent to produce high reward behaviour on the new task by reweighting a retrieved sub-trajectory by the total return achieved over the episode it appears in, i.e., $s_{\text{u}}(\tau_{\text{ret}}, \tau_{\text{in}}) = \sum_{i=0}^{T} r_i$. We apply this reweighting to the retrieved experiences in $\mathcal{R}$ and select the top-k elements by:

$$\mathcal{S} = \arg\max_{\boldsymbol{k}, \tau_{\text{ret}}\in\mathcal{R}}^{k} s_{\text{ret}}(\boldsymbol{k}, \boldsymbol{q}, \tau_{\text{ret}}), \tag{5}$$

where we normalize both scores to be in the range $[0, 1]$, such that they contribute equally to the final weight. Our reweighting mechanism is illustrated in Figure 2.

Algorithm 1 In-context Learning with RA-DT

Input: DT π_θ, embed model g, episodes N, episode len T, context len C, `retrieve`, `reweight`.

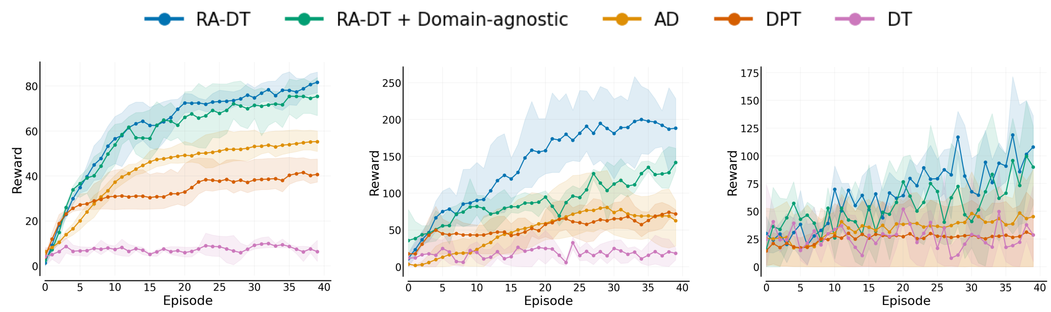
$\mathcal{I} \leftarrow \emptyset$ ▷ Inititalize index
for $1 \dots N$ **do**
  $s, \tau \leftarrow$ `env.reset()`$, \emptyset$
  for $t = 1 \dots T$ **do**
    $\boldsymbol{q} = g(\tau_{t-C:t})$ ▷ Construct query
    $\mathcal{R} \leftarrow$ `retrieve`$(\boldsymbol{q}, \mathcal{I})$ ▷ Top-l trjs, Eq. 3
    $\mathcal{S} \leftarrow$ `reweight`$(\mathcal{R})$ ▷ Top-k, Eq. 4, 5
    $a \sim \pi_\theta(a \mid \tau_{t-C:t}, \{\tau_{\text{ret}} \in \mathcal{S}\})$ ▷ Predict
    $s', r \leftarrow$ `env.step`(a)
    $\tau \leftarrow \tau \cup (s, a, r)$ ▷ Append transition to τ
    $s \leftarrow s'$
  end for
  $\mathcal{I} \leftarrow \mathcal{I} \cup \tau$ ▷ Add trajectory τ to index $\mathcal{I}$
end for

3.2.4 Incorporating Retrieved Experiences

After reweighting, the set $\mathcal{S}$ contains sub-trajectories that are both important and relevant for the current input τ_{in} to the DT π_θ. To incorporate the retrieved experiences in the DT, we interleave it with cross-attention layers (CA) after every self-attention (SA) layer. The retrieved sub-trajectories are encoded by separate embedding layers for each token type (state/action/reward/RTG) and then passed to the CA layers. Thus, our RA-DT predicts actions a_t given input trajectory and retrieved trajectory by:

$$a_t \sim \pi_\theta(a_t \mid \tau_{\text{in}}, \{\tau_{\text{ret}} \in \mathcal{S}\}). \tag{6}$$

In Algorithm 1, we show the pseudocode for in-context RL with RA-DT at *inference* time. In addition, we show RA-DT at *training* time in Algorithm 2 of Appendix C.4.



(a) Dark-Room 10×10 (b) Dark-Room 20×20 (c) Dark-Room 40×20

Figure 3: ICL performance on **Dark-Room (a)** 10×10, **(b)** 20×20, **(c)** 40×20 at end of training (100K steps). We evaluate each agent for 40 episodes on each of the 20 **evaluation tasks** and report mean reward (+ 95% CI) over 3 seeds.

4 Experiments

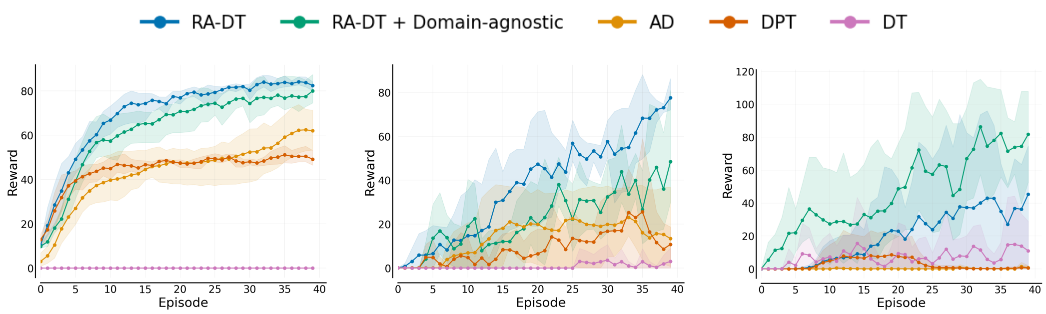
We evaluate the ICL abilities of RA-DT on grid-world environments used in prior works, namely Dark-Room (see Section 4.1), Dark Key-Door (Section 4.2), and MazeRunner (Section 4.3) [Laskin et al., 2022; Lee et al., 2022; Grigsby et al., 2023], with increasingly larger grid-sizes, resulting in longer episodes. Moreover, we evaluate RA-DT on two robotic benchmarks (Meta-World and DMControl, Section 4.4) and procedurally-generated video games (Procgen, Section 4.5).

Across experiments, we report performances for two variants of **RA-DT**. The first variant leverages a domain-specific embedding model for retrieval, specifically a DT trained on the same domain. The second variant (**RA-DT + Domain-agnostic**) makes use of the FH mechanism in combination with BERT [Devlin et al., 2019] as the pre-trained LM. Consequently, this variant of RA-DT does not require any domain-specific pre-training of the embedding model. We compare RA-DT against the vanilla **DT** and two established in-context RL methods, namely Algorithm Distillation [Laskin et al., 2022, **AD**] and Decision Pre-trained Transformer [Lee et al., 2023, **DPT**]. Following, Agarwal et al. [2021] we report the mean across tasks and 95% confidence intervals over 3 seeds. We use a context length equivalent to two episodes (from 200 up to 2000 timesteps) for AD, DPT and DT. For RA-DT, we use a considerably shorter context length of 50 transitions, unless mentioned otherwise. On grid-worlds, we train all methods for 100K steps and evaluate after every 25K steps. Similarly, we train for 200K steps and evaluate after every 50K steps for Meta-World, DMControl and Procgen. All grid-worlds and Procgen exhibit discrete actions and consequently, we train all methods via the cross-entropy loss to predict the next actions. On Meta-World and DMControl, we train all method using the mean-squared error loss to predict continuous actions. Following Laskin et al. [2022] and Lee et al. [2023], our primary evaluation criterion is performance improvement during ICL trials. After training, the agent interacts with the environment for a fixed amount of episodes, each of which is considered a single trial. Upon completion of an ICL trial, the respective episode is stored in the vector index. We provide further training and implementation details in Appendix C.

4.1 Dark-Room

Experiment Setup. Dark-Room is commonly used in prior work on in-context RL [Laskin et al., 2022; Lee et al., 2023]. The agent is located in an empty room, observes only its x-y coordinates, and has to navigate to an invisible goal state ($|\mathcal{S}| = 2$, $|\mathcal{A}| = 5$, see Figure 9). A reward of +1 is obtained in every step the agent is located in the goal state. Because of partial observability, it must leverage memory of previous episodes to find the goal. We conduct experiments on three different grid sizes, namely 10×10, 20×20, and 40×20, and corresponding episode lengths of 100, 200 and 800, respectively. We designate 80 and 20 randomly assigned goals as train and evaluation locations, respectively, as in Lee et al. [2023]. We use Proximal Policy Optimization (PPO) [Schulman et al., 2017] to generate 100K transitions per goal for 10×10 and 20×20 grids and 200K for 40×20 (see Figure 7 for single task expert scores). During evaluation, the agent interacts with the environment for 40 ICL trials, and we report the scores at the last evaluation step (100K). We provide additional details on the environment, the generated data, and the training procedure in Appendix B.1 and C.

Results. In Figure 3, we show the ICL performances on the 20 hold-out tasks for all considered methods on Dark-Room **(a)**10×10, **(b)** 20×20, and **(c)** 40×20. In addition, we present the ICL curves on the training tasks and the learning curves across the entire training period in Figures 14 and 15 in Appendix D.1. Overall, we observe that RA-DT attains the highest average rewards on all 3 grid-sizes at the end of the 40 ICL-trials. On 10×10, RA-DT obtains near-optimal performance scores both with the domain-specific and domain-agnostic embedding model. The vanilla DT does not exhibit any performance improvement across trials. This indicates the improvement in performance for RA-DT can be attributed to the retrieval component. Furthermore, RA-DT outperforms AD and DPT without keeping entire episodes in its context window. Similarly, RA-DT outperforms all baselines on the 20×20 and 40×20 grids. While RA-DT successfully improves in-context, the baselines exhibit only little learning progress over the ICL trials, especially for larger grid sizes. However, the final performance scores for 20×20 and 40×20 are not optimal. With increasing grid size, discovering the goal requires systematic exploration in combination with targeted exploitation. Therefore, we conduct a qualitative analysis on the exploration behaviour of RA-DT. We find that RA-DT develops strategies to imitate a given successful context (see Figure 16), and avoids low-reward routes given an unsuccessful context (see Figure 17).



(a) Dark Key-Door 10×10 (b) Dark Key-Door 20×20 (c) Dark Key-Door 40×20

Figure 4: ICL performance on **Dark Key-Door (a)** 10×10, **(b)** 20×20, **(c)** 40×20 at end of training (100K steps). We evaluate each agent for 40 episodes on each of the 20 **evaluation tasks** and report mean reward (+ 95% CI) over 3 seeds.

4.2 Dark Key-Door

Experiment Setup. In Dark Key-Door, the agent is located in a room with two invisible objects: a key and a door. The agent has to pick up the invisible key, then navigate to the door. Because of the presence of two key events, the task-space is combinatorial in the number of grid-cells ($100^2 = 10000$ possible tasks for 10×10) and is therefore considered more difficult. A reward of +1 is obtained once for picking up the key and for every step the agent stands on the door grid-cell after it collected the key. We retain the same experiment setup as in Section 4.1 and provide further details in Appendix B.1 (also see Figure 8 for single-task expert scores).

Results. On 10×10 and 20×20, RA-DT outperforms baselines, with the performance ranking remaining the same as on Dark-Room (see Figure 4). Surprisingly, domain-agnostic RA-DT outperforms its domain-specific counterpart on 40×20, which demonstrates that the domain-agnostic embedding model is a promising alternative. This result indicates that RA-DT can successfully handle environments with more than one key event, even with shorter observed context.

4.3 Maze-Runner

Experiment Setup. Maze-Runner was introduced by Grigsby et al. [2023] and inspired by Pasukonis et al. [2022]. The agent is located in a procedurally-generated 15×15 maze (see Figure 10), observes continuous Lidar-like depth representations of states, and has to navigate to one, two, or three goal locations in the correct order ($|\mathcal{S}| = 6$,$|\mathcal{A}| = 4$). A reward of +1 is obtained when reaching a goal location. Episodes last for a maximum of 400 steps, or terminate early if all goal locations have been visited. Similar to Dark-Room, we use PPO to generate 100K environment interactions for 100 procedurally-generated mazes. We train all methods on a multi-task dataset that comprises trajectories from 100 mazes, evaluate on 20 unseen mazes, and report performance over 30 ICL trials. We give further details on the environment, the dataset, and the experiment setup in Appendix B.2 and D.2.

Results. We find that RA-DT considerably outperforms all baselines in terms of final performance (see Figure 5). Surprisingly, RA-DT is the only method to improve over the course of the 30 ICL trials. However, we observe a considerable performance gap between train mazes and test mazes (0.65 vs. 0.4 reward, see Figure 20), indicating that solving unseen mazes requires an enhanced ability to generalize and learn from previous trials.

4.4 META-WORLD & DMCONTROL

Experiment Setup. Next, we evaluate RA-DT on two multi-task robotics benchmarks, Meta-World [Yu et al., 2020b] and DMControl [Tassa et al., 2018]. States and actions in both benchmarks are multidimensional continuous vectors. While the state and action space in Meta-World remain constant across all tasks ($|\mathcal{S}| = 39$, $|\mathcal{A}| = 6$), they vary considerably in DMControl ($3 \leq |\mathcal{S}| \leq 24$, $1 \leq |\mathcal{A}| \leq 6$). Episodes last for 200 and 1000 steps in Meta-World and DMControl, respectively. We leverage the datasets released by Schmied et al. [2024]. For Meta-World, we pre-train a multi-task policy on 45 of the 50 tasks (ML45, 90M transitions in total) and evaluate on the 5 remaining tasks (ML5). Similarly, on DMControl, we pre-train on 11 tasks (DMC11, 11M transitions in total) and evaluate on 5 unseen tasks (DMC5). We provide further details on the environments, datasets, and experiment setup in Appendices B.3 and D.3, and B.4 and D.4 for Meta-World and DMControl, respectively.

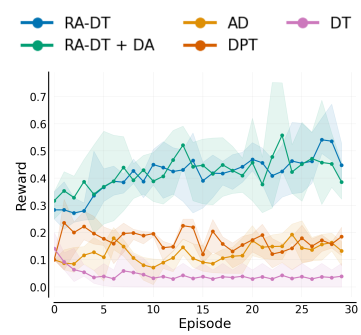


Figure 5: ICL on **MazeRunner**. We evaluate over 30 ICL trials and report the mean reward (+ 95% CI) over 3 seeds.

Results. We present the learning curves and corresponding ICL curves for Meta-World and DMControl in Figure 22 and 23, and Figures 24 and 25 in Appendix D, respectively. In addition, we provide the raw and data-normalized scores in Tables 3 and 4, respectively. On both benchmarks, we find that RA-DT attains considerably higher scores on unseen evaluation tasks, but slightly lower average scores across training tasks compared to DT. However, these performance gains on evaluation tasks are not reflected in improved ICL performance. In fact, we only observe slight in-context improvement on training tasks, but not on holdout tasks for any of the considered methods.

4.5 PROCGEN

Experiment Setup. Finally, we conduct experiments on Procgen [Cobbe et al., 2020], a benchmark consisting of 16 procedurally-generated video games, designed to test the generalization abilities of RL agents. The procedural generation in Procgen is controlled by setting an environment seed, which results in visually diverse observations for the same underlying task (see `starpilot`-example in Figure 12). In Procgen, the agent receives image-based inputs ($|\mathcal{S}| = 3\times64\times64$). All 16 tasks share a discrete action space ($|\mathcal{A}| = 15$). Rewards are either dense or sparse depending on the environment.

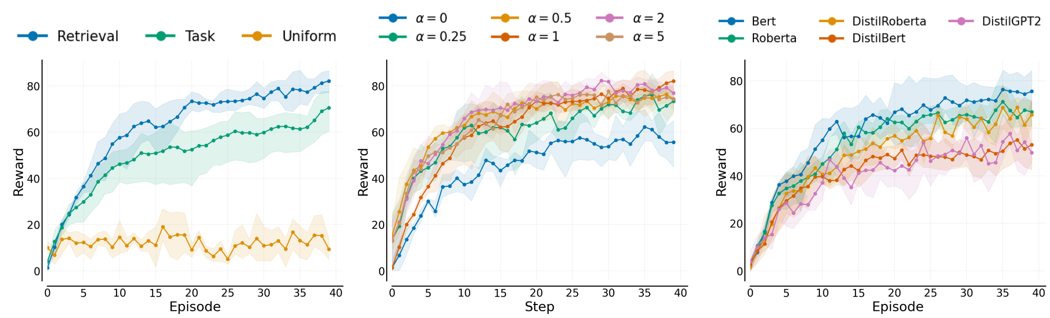
We follow Raparthy et al. [2023] and use 12 tasks for training (PG12) and 4 tasks for evaluation (PG4). First, we generate datasets by training task-specific PPO agents for 25M timesteps on 200 environment seeds per task in `easy` difficulty. Then, we pre-train a multi-task policy on the PG12 datasets (24M transitions in total, 2M per task). We leverage the procedural generation of Procgen and evaluate all models in three settings: *training tasks - seen* (PG12-Seen), *training tasks - unseen* (PG12-Unseen), and *evaluation tasks - unseen* (PG4). Additional details on the generated datasets and our environment setup are available in Appendices B.5 and D.5.

Results. Similar to our results on Meta-World and DMControl, we find that RA-DT improves average performance scores across all three settings compared to the baselines (see Figure 26 and Tables 5, 6, 7 in Appendix D.5), but no method exhibits in-context improvement during evaluation (Figure 27). We further discuss our negative results on Procgen, Meta-World, and DMControl in Section 5.

4.6 Ablations

To better understand the effect of learning with retrieval, we present a number of ablation studies on essential components in RA-DT conducted on Dark-Room 10×10 (more details in Appendix E).

Retrieval outperforms sampling of experiences. To investigate the effect of learning with retrieved context, we substitute retrieval with random sampling, either over all tasks, or from the same task (see Figure 6a). We find that training with retrieval outperforms both sampling variants, highlighting the benefit of training with retrieval to improve ICL abilities. We hypothesise this is because retrieval constructs bursty sequences, which was found to be important for ICL [Chan et al., 2022].



(a) (b) (c)

Figure 6: Ablations on important components in RA-DT conducted on Dark-Room 10×10. We show **(a)** the effect of training with retrieval vs. sampling, **(b)** a sensitivity analysis on α as used in the re-weighting mechanism during training, and **(c)** the effect of leveraging different LMs as pre-trained embedding models for domain-agnostic retrieval.

Reweighting Experiences. RA-DT reweights a sub-trajectory by its *relevance* and *utility* score. By default, we use task-based reweighting during training. In Figure 28, we compare against alternatives, such as reweighting by return. Indeed, we find that task-based reweighting is critical for high performance, because it ensures that retrieved experiences are useful for predicting the next action.

Sensitivity of Reweighting. We conduct a sensitivity analysis on α used in the reweighting mechanism (see Equation 4). In Figure 6b, we find that RA-DT performs well for a range of values for α used during training, but performance declines if no re-weighting is employed ($\alpha = 0$). We perform the same analysis for α during evaluation in Figure 29.

Effect of Retrieval Regularization. We evaluate with three retrieval regularization strategies to mitigate the effect of copying the context: deduplication, similarity cut-off, and query dropout. To evaluate their impact on ICL performance, we systematically removed each one from RA-DT (see Figure 30). We found the combination of all three to be effective and add them to our pipeline.

Different LMs for domain-agnostic RA-DT. Finally, we investigate how strongly domain-agnostic RA-DT is influenced by the choice of pre-trained LM for the embedding model. We compare our default choice BERT against other smaller/larger LMs (see Figure 36). We found that BERT performs best and performance decreases with smaller models.

Effect of Retrieval on Training/Inference Efficiency. Retrieval-augmentation adds computational overhead to the training/inference pipeline due to the cost of embedding the query and searching for similar experiences. However, we find that RA-DT results in significantly faster training times because of shorter context length (up to $7\times$ see Appendix E.7). At inference-time, RA-DT is slightly slower compared to baselines when retrieving at every step, but exhibits similar inference speeds when retrieving less frequently (see Appendix E.8). Importantly, the retrieval mechanism in RA-DT enables access to the entirety of the experiences collected across all ICL trials with small additional cost. The ability to access a broader range of experiences may be a reason for its enhanced performance.

For additional ablations on RA-DT and on our baselines, we refer to Appendix E.

5 DISCUSSION

In this section, we highlight current challenges of RA-DT and other offline in-context RL methods.

Memory-Exploitation vs. Meta-learning Abilities. Current *offline* in-context RL methods are predominantly evaluated on contextual bandits or grid-worlds, such as Dark-Room [Laskin et al., 2022; Lee et al., 2023; Lin et al., 2023; Sinii et al., 2023; Huang et al., 2024], which can only be solved by leveraging the context. However, it remains unclear to what extent the agent learns to learn in-context or simply copies from its context. Further, in our experiments on fully-observable environments (MetaWorld, DMControl, and Procgen), we did not observe ICL behaviour (see Appendices D.3, D.4, D.5). Therefore, it is necessary that future research on in-context RL disentangles the effects of memory and meta-learning abilities, similar to memory and credit-assignment [Ni et al., 2024].

Challenges of Next-Action Prediction. Most in-context RL methods learn from offline datasets via next-action prediction and causal sequence modelling objectives. As such, they cannot learn to infer the utility of an action, and thus, distinguish between positive and negative examples. This can induce delusions, which lead to repetitions of suboptimal actions and copying behaviour [Ortega et al., 2021] (see Figure 19 for examples on Dark-Room). In contrast, *online* in-context RL methods have shown promising adaptation abilities [Team et al., 2023; Grigsby et al., 2023; Lu et al., 2024]. A similar trend has been observed in online meta-RL methods [Melo, 2022; Shala et al., 2024]. Consequently, a potential remedy to this problem is to train a value function to learn the utility of an action. However, this is usually not straightforward and requires constrained optimization objectives [Zanette et al., 2021; Kumar et al., 2020]. Therefore we leave this approach to future work.

Conditioning Strategies in RL. In LLMs, applying sophisticated conditioning strategies is important to improve ICL abilities [Wei et al., 2022; Yao et al., 2024; Agarwal et al., 2024]. Even though RTG-conditioning [Chen et al., 2021], and chain-of-hindsight [Liu & Abbeel, 2023] have shown promise for generating high reward behaviour in DTs, the broader landscape for conditioning strategies for in-context RL remains under-explored. Therefore, we believe that systematically investigating conditioning methods for in-context RL is a fruitful direction for future research.

Diversity of the Pre-training Distribution. The diversity and scale of the pre-training dataset may significantly affect the emergence of ICL. In our experiments, we pre-train on a relatively small set of tasks. Our results on gridworlds suggest that this is sufficient for ICL to emerge on simple environments. However, on more complex environments, the unseen tasks can be considered out-of-distribution and higher pre-training diversity may be necessary for ICL to emerge. It remains unclear how much diversity is required to elicit in-context RL, and if existing large-scale agents exhibit ICL [Reed et al., 2022; Raad et al., 2024]. One promising approach is to expand the pre-training diversity through learned interactive simulations [Yang et al., 2023a; Bruce et al., 2024].

6 CONCLUSION

Existing in-context RL methods keep entire episodes in their context window, which is challenging as RL environments are typically characterized by long episodes and sparse rewards. To address this challenge, we introduce RA-DT, which employs an external memory mechanism to store past experiences and to retrieve experiences relevant for the current situation. RA-DT outperforms baselines on grid-worlds, while using only a fraction of their context length. While RA-DT improves average performance on holdout tasks on complex environments, it struggles to exhibit ICL, along with other in-context RL methods. Consequently, we illuminate the current limitations of in-context RL methods and discuss future directions. Finally, we release our datasets for Dark-Room, Dark Key-Door, MazeRunner, and Procgen, to facilitate future research on in-context RL.

Future Work. Besides the general directions discussed in Section 5, we highlight a number of concrete approaches to extend RA-DT. While we focus on ICL without relying on expert demonstrations, pre-filling the external memory with demonstrations may enable RA-DT to perform more complex tasks. This may be effective for robotics applications, where expert demonstrations are easy to obtain. Furthermore, end-to-end training of the retrieval component in RA-DT, similar to [Izacard et al., 2022], may result in more precise context retrieval and enhanced down-stream performance. Finally, we envision that modern recurrent architectures [Bulatov et al., 2022; Gu & Dao, 2023; Beck et al., 2024] as policy backbones may benefit RA-DT by maintaining hidden states across many episodes.

REFERENCES

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.

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Stephanie Chan, Ankesh Anand, Zaheer Abbas, Azade Nova, John D Co-Reyes, Eric Chu, et al. Many-shot in-context learning. *arXiv preprint arXiv:2404.11018*, 2024.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: return decomposition for delayed rewards. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 13544–13555, 2019.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack W. Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *CoRR*, abs/1606.04460, 2016.

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*, pp. 2206–2240. PMLR, 2022.

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.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. *arXiv preprint arXiv:2402.15391*, 2024.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

Stephanie Chan, Adam Santoro, Andrew K. Lampinen, Jane Wang, Aaditya Singh, Pierre H. Richemond, James L. McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Rudi D'Hooge and Peter P De Deyn. Applications of the morris water maze in the study of learning and memory. *Brain research reviews*, 36(1):60–90, 2001.

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*, pp. 1407–1416. PMLR, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Sebastian Flennerhag, Andrei A Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*, 2019.

Anirudh Goyal, Abram Friesen, Andrea Banino, Theophane Weber, Nan Rosemary Ke, Adria Puigdomenech Badia, Arthur Guez, Mehdi Mirza, Peter C Humphreys, Ksenia Konyushova, et al. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pp. 7740–7765. PMLR, 2022.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

Jake Grigsby, Linxi Fan, and Yuke Zhu. Amago: Scalable in-context reinforcement learning for adaptive agents. *arXiv preprint arXiv:2310.09971*, 2023.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *Artificial Neural Networks—ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*, pp. 87–94. Springer, 2001.

Ziniu Hu, Ahmet Iscen, Chen Sun, Zirui Wang, Kai-Wei Chang, Yizhou Sun, Cordelia Schmid, David A. Ross, and Alireza Fathi. Reveal: Retrieval-augmented visual-language pre-training with multi-source multimodal knowledge memory. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*, pp. 23369–23379. IEEE, 2023. doi: 10.1109/CVPR52729.2023.02238.

Sili Huang, Jifeng Hu, Hechang Chen, Lichao Sun, and Bo Yang. In-context decision transformer: Reinforcement learning via hierarchical chain-of-thought. *arXiv preprint arXiv:2405.20692*, 2024.

Peter Humphreys, Arthur Guez, Olivier Tieleman, Laurent Sifre, Théophane Weber, and Timothy Lillicrap. Large-scale retrieval for reinforcement learning. *Advances in Neural Information Processing Systems*, 35:20092–20104, 2022.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.

Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

Louis Kirsch, James Harrison, Jascha Sohl-Dickstein, and Luke Metz. General-purpose in-context learning by meta-learning transformers. *arXiv preprint arXiv:2212.04458*, 2022.

Louis Kirsch, James Harrison, C Freeman, Jascha Sohl-Dickstein, and Jürgen Schmidhuber. Towards general-purpose in-context learning agents. In *NeurIPS 2023 Workshop on Generalization in Planning*, 2023.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Dharshan Kumaran, Demis Hassabis, and James L. McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20: 512–534, 2016.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

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.

Jonathan N Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *arXiv preprint arXiv:2306.14892*, 2023.

Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

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.

Licong Lin, Yu Bai, and Song Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. *arXiv preprint arXiv:2310.08566*, 2023.

Hao Liu and Pieter Abbeel. Emergent agentic transformer from chain of hindsight experience. *arXiv preprint arXiv:2305.16554*, 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

James L. Mcclelland, Bruce L. Mcnaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102:419–457, 1995.

Luckeciano C. Melo. Transformers are meta-reinforcement learners. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 15340–15359. PMLR, 2022.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=B1DmUzWAW.

Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. When do transformers shine in rl? decoupling memory from credit assignment. *Advances in Neural Information Processing Systems*, 36, 2024.

Pedro A. Ortega, Jane X. Wang, Mark Rowland, Tim Genewein, Zeb Kurth-Nelson, Razvan Pascanu, Nicolas Heess, Joel Veness, Alexander Pritzel, Pablo Sprechmann, Siddhant M. Jayakumar, Tom McGrath, Kevin J. Miller, Mohammad Gheshlaghi Azar, Ian Osband, Neil C. Rabinowitz, András György, Silvia Chiappa, Simon Osindero, Yee Whye Teh, Hado van Hasselt, Nando de Freitas, Matthew M. Botvinick, and Shane Legg. Meta-learning of sequential strategies. *CoRR*, abs/1905.03030, 2019.

Pedro A Ortega, Markus Kunesch, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Joel Veness, Jonas Buchli, Jonas Degrave, Bilal Piot, Julien Perolat, et al. Shaking the foundations: delusions in sequence models for interaction and control. *arXiv preprint arXiv:2110.10819*, 2021.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pp. 17156–17185. PMLR, 2022.

Fabian Paischer, Thomas Adler, Markus Hofmarcher, and Sepp Hochreiter. Semantic HELM: an interpretable memory for reinforcement learning. *CoRR*, abs/2306.09312, 2023. doi: 10.48550/arXiv.2306.09312.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Jurgis Pasukonis, Timothy Lillicrap, and Danijar Hafner. Evaluating long-term memory in 3d mazes. *arXiv preprint arXiv:2210.13383*, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Vihang Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M. Blies, Johannes Brandstetter, José Antonio Arjona-Medina, and Sepp Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17531–17572. PMLR, 2022.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2827–2836. PMLR, 2017.

Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

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.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *arXiv preprint arXiv:2302.00083*, 2023.

Rita Ramos, Bruno Martins, Desmond Elliott, and Yova Kementchedjhieva. Smallcap: Lightweight image captioning prompted with retrieval augmentation. *CoRR*, abs/2209.15323, 2022. doi: 10.48550/arXiv.2209.15323.

Sharath Chandra Raparthy, Eric Hambro, Robert Kirk, Mikael Henaff, and Roberta Raileanu. Generalization to new sequential decision making tasks with in-context learning, 2023.

Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *CoRR*, abs/2205.06175, 2022. doi: 10.48550/arXiv.2205.06175.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987.

Dominik Schmidt and Thomas Schmied. Fast and data-efficient training of rainbow: an experimental study on atari. *arXiv preprint arXiv:2111.10247*, 2021.

Thomas Schmied, Markus Hofmarcher, Fabian Paischer, Razvan Pascanu, and Sepp Hochreiter. Learning to modulate pre-trained models in rl. *Advances in Neural Information Processing Systems*, 36, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pp. 30365–30380. PMLR, 2023.

Kajetan Schweighofer, Marius-constantin Dinu, Andreas Radler, Markus Hofmarcher, Vihang Prakash Patil, Angela Bitto-Nemling, Hamid Eghbal-zadeh, and Sepp Hochreiter. A dataset perspective on offline reinforcement learning. In *Conference on Lifelong Learning Agents*, pp. 470–517. PMLR, 2022.

Philipp Seidl, Philipp Renz, Natalia Dyubankova, Paulo Neves, Jonas Verhoeven, Jorg K Wegner, Marwin Segler, Sepp Hochreiter, and Gunter Klambauer. Improving few-and zero-shot reaction template prediction using modern hopfield networks. *Journal of chemical information and modeling*, 62(9):2111–2120, 2022.

Gresa Shala, André Biedenkapp, and Josif Grabocka. Hierarchical transformers are efficient meta-reinforcement learners. *CoRR*, abs/2402.06402, 2024. doi: 10.48550/ARXIV.2402.06402. URL https://doi.org/10.48550/arXiv.2402.06402.

Viacheslav Sinii, Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, and Sergey Kolesnikov. In-context reinforcement learning for variable action spaces. *arXiv preprint arXiv:2312.13327*, 2023.

Pablo Sprechmann, Siddhant M. Jayakumar, Jack W. Rae, Alexander Pritzel, Adrià Puigdomènech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, October 2012. doi: 10.1109/IROS.2012.6386109.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack W. Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Jimenez Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matthew M. Botvinick, Demis Hassabis, and Timothy P. Lillicrap. Unsupervised predictive memory in a goal-directed agent. *CoRR*, abs/1803.10760, 2018.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks, 2015.

Michael Widrich, Markus Hofmarcher, Vihang Prakash Patil, Angela Bitto-Nemling, and Sepp Hochreiter. Modern hopfield networks for return decomposition for delayed rewards. In *Deep RL Workshop NeurIPS 2021*, 2021.

Maciej Wolczyk, Michal Zajkac, Razvan Pascanu, Lukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 34:28496–28510, 2021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics.

Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 2023a.

Zhuolin Yang, Wei Ping, Zihan Liu, Vijay Korthikanti, Weili Nie, De-An Huang, Linxi Fan, Zhiding Yu, Shiyi Lan, Bo Li, Ming-Yu Liu, Yuke Zhu, Mohammad Shoeybi, Bryan Catanzaro, Chaowei Xiao, and Anima Anandkumar. Re-vilm: Retrieval-augmented visual language model for zero and few-shot image captioning. *CoRR*, abs/2302.04858, 2023b. doi: 10.48550/arXiv.2302.04858.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Richard James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen-Tau Yih. Retrieval-augmented multimodal language modeling. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 39755–39769. PMLR, 2023.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020a.

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*, pp. 1094–1100. PMLR, 2020b.

Andrea Zanette, Martin J. Wainwright, and Emma Brunskill. Provable benefits of actor-critic methods for offline reinforcement learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp.

13626–13640, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/713fd63d76c8a57b16fc433fb4ae718a-Abstract.html.

Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *CoRR*, abs/1505.00521, 2015.

K.J Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(65)90154-X.

Appendix

Contents

A Ethics Statement & Reproducibility

In recent years, there has been a trend in RL towards large-scale multi-task models that leverage offline pre-training. In this work, we broadly aim at building agents that can learn new tasks via ICL without the need for re-training or fine-tuning. Our goal is to reduce the need to provide entire past episodes in the agent's context, by augmenting the agent with an external memory in combination with a retrieval component, similar to RAG in LLMs. We believe that multi-task agents of the near future will be able to perform a broad range of tasks, and that these agents will greatly benefit from RAG as used in RA-DT. The external memory component can enable agents to leverage information from in its own distant past or experiences from other agents. Such agents could have an immense impact on the global economy (e.g., as a source of inexpensive labour). As such, they do not come without risks and the potential for misuse. While we believe that our work can significantly impact the positive use of future agents, it is essential to ensure responsible deployment of future technologies.

Upon publication, we will open-source the code-base used for our experiments, and release the datasets we generated. In addition, we provide further information on the environments/datasets, implementation including hyperparameter tables, and on our experiments in Appendices B, C, D, respectively.

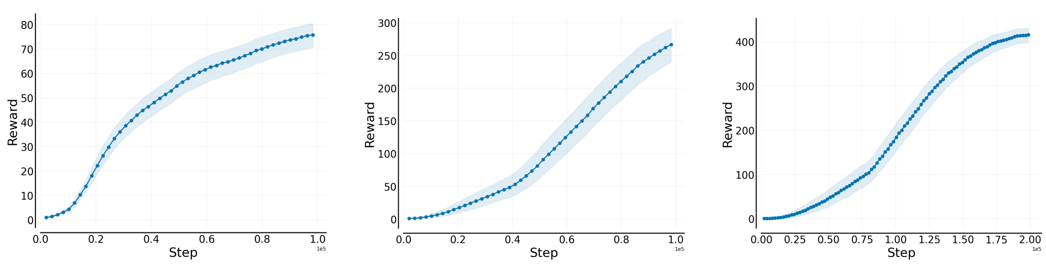
B Environments & Datasets

B.1 Dark-Room and Dark Key-Door

The Dark-Room environment is modelled after Morris-Watermaze, a classic experiment in behavioural neuroscience for studying spatial memory and learning in animals [D'Hooge & De Deyn, 2001]. We design our Dark-Room and Dark Key-Door environments in Minihack [Samvelyan et al., 2021], which is based on the NetHack Learning Environment [Küttler et al., 2020]. We construct grids of dimensions 10×10, 20×20 and 40×20, as depicted in Figure 9. With increasing grid sizes, the task of locating the goal becomes harder as the number of possible positions in the grid grows (100, 400, 800). Therefore, we set the number of interaction steps per environment equal to the number of grid cells. Consequently, larger grids results in longer episodes and thus context lengths (e.g., 2400 for AD). The agent observes its own x-y position on the grid and can perform one of 5 actions at every interaction step (up, down, left, right, stay). Episodes start in the top left corner (0,0) and the agent is reset to the start position after every episode.

In **Dark-Room**, the agent has to navigate to a randomly placed and invisible goal position. Therefore, the task space in Dark-Room environments is equal to the number of grid-cells (i.e., 100 for 10×10). The agent receives a reward for +1 for every step in the episode it is located in the goal position and 0 otherwise. As there are as many grid-cells as episode steps, the optimal strategy for solving the Dark-Room task is to use the first episode to visit every cell to find the hidden goal location. Once found, this knowledge can be exploited in upcoming trials.

In contrast, in **Dark Key-Door**, there are two objects: a key and a goal state. Similar to Dark-Room, the key and goal position are randomly placed on the grid. The agent has to first pick up the invisible key and then find the invisible goal. Due to the presence of the two key events (picking up the key, finding the goal), the task space is combinatorial in the number of grid-cells (i.e., $100^2 = 10000$ for 10×10). This makes the Dark Key-Door more challenging than the Dark-Room task, especially as the grid-size becomes larger.



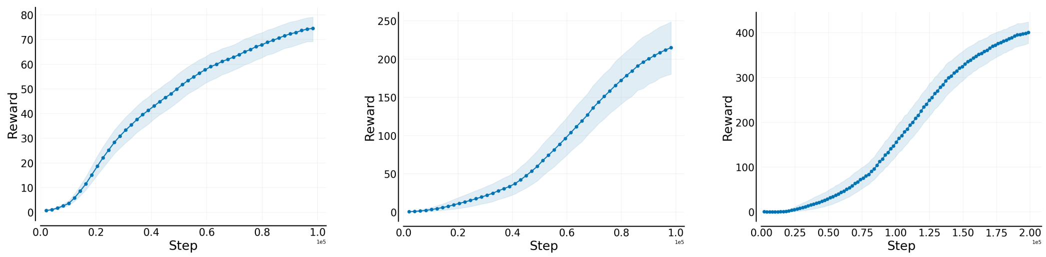
(a) Dark-Room 10×10 (b) Dark-Room 20×20 (c) Dark-Room 40×20

Figure 7: Average performances of the **source algorithm**, PPO, on 80 train tasks for Dark-Room **(a)** 10×10, **(b)** 20×20, and **(c)** 40×20. For (a), (b) we train PPO on individual tasks for 100K environment steps. For (c), we train for 200K environment steps to take the longer episode lengths into account. We evaluate the agents after every 10K steps. Curves show the mean reward achieved (+ 95% CI) across the 80 train tasks.

Training Dataset. For both Dark-Room and Dark Key-Door, we generate training datasets for 80 randomly assigned goals or key-goal combinations. We use PPO [Schulman et al., 2017] to generate 100K environment transitions per goal location for 10×10 and 20×20 grids and 200K environment transitions for the largest grid. Therefore, the total number of transitions across datasets is 8M for 10×10 and 20×20 grids and 16M for 40×20.

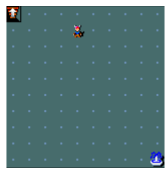
We train PPO with standard hyperparameter settings in `stable-baselines3` [Raffin et al., 2021] using a learning rate of $3e^{-4}$, batch size of 64, number of steps between updates of 2048, number of update epochs 10 and entropy coefficient of 0.01. For 20×20 and 40×20 grids, we increase the number of update epochs to 30 and the entropy coefficient of to 0.1 for 40×20. We store all generated transitions of PPO for our datasets. Consequently, the final datasets contain a mixture of suboptimal or exploratory, and optimal or exploitative behaviour.

Source Algorithm Performance. We show average learning curves across all task-specific PPO agents on the 80 training tasks for all grid-sizes in Figures 7 and 8 for Dark-Room and Dark Key-Door, respectively. For the 10×10 grids, the average performance converges towards optimal performance. However, on the larger grid sizes, the performances are below the optimum. This is because it takes the agent longer to discover and collect successful episodes by initially random environment interaction as the grids become larger.



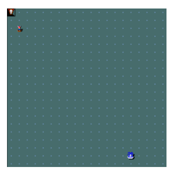
(a) Dark Key-Door 10×10 (b) Dark Key-Door 20×20 (c) Dark Key-Door 40×20

Figure 8: Average performances of the **source algorithm**, PPO, on 80 train tasks for **Dark Key-Door (a)** 10×10, **(b)** 20×20, and **(c)** 40×20. For (a), (b) we train PPO on individual tasks for 100K environment steps. For (c), we train for 200K environment steps. We evaluate the agents after every 10K steps. Curves show the mean reward achieved (+ 95% CI) across the 80 train tasks.

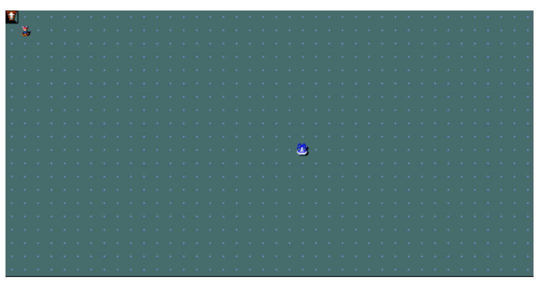


(a) Room 10×10

(b) Key-Door 10×10



(c) Room 20×20

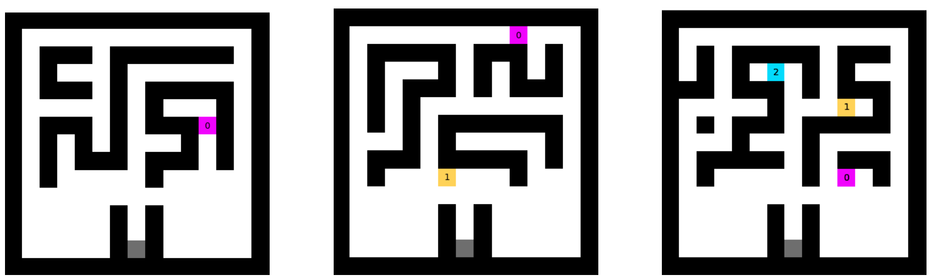


(d) Room 40×20

Figure 9: Mini-grid environments. In **Dark-Room**, the agent is located in a room and has to navigate to an invisible goal location. We use grid-sizes **(a)** 10×10, **(b)** 20×20 and **(c)** 40×20 for our experiments. In **(b) Dark-KeyDoor**, the agent has to pick up an invisible key, then navigate to the invisible goal location. Agents only observe their current x-y coordinate on the grid. Reward of +1 is obtained in every step the agent is situated in the goal state, +1 for picking up the key.

B.2 MazeRunner

MazeRunner was introduced by [Grigsby et al., 2023] and inspired by the Memory Maze environment [Pasukonis et al., 2022]. The agent is located in a 15×15 procedurally-generated maze and has to navigate to a sequence of one, two, or three goal locations in the right order (see Figure 10). Similar to Dark-Room environments, MazeRunner is partially observable and exhibits sparse rewards. The agent observes a Lidar-like 6-dimensional representation of the state that contains 4 continuous values that measure the distance from the agent's location to the nearest wall, and the x-y coordinates of the agent's position in the grid. The action-space is 4-dimensional (up, down, left, right). A reward of +1 is obtained when reaching the currently active goal state in the goal sequence. Therefore, the total achievable reward is equal to the number of goal states. Episodes last for a maximum of 400 steps or terminate early, if all goal locations have been reached. After every episode, the agent (gray box in Figure 10) is reset to the origin location. During evaluation, we allow for 30 ICL trials, which amounts to 12K environment steps in total.



(a) One goal (b) Two goals (c) Three goals

Figure 10: Maze-Runner environments introduced by Grigsby et al. [2023]. In **Maze-Runner**, the agent is located in a procedurally generated 15×15 maze and has to navigate to **(a)** one, **(b)** two or **(c)** goal locations in pre-specified order. The agent receives a reward of +1 for reaching a goal. Episodes last for a maximum of 400 steps, or terminate early if all goal locations have been visited.

Training Dataset. The procedural-generation of the maze and selection of the number of goals is controlled by setting the environment seed. We use PPO to generate 100K environment interactions for 100 procedurally-generated mazes, and record the entire replay buffer, which amounts to 10M transitions in total. We found it necessary, to equip the task-specific PPO agents with an LSTM [Hochreiter & Schmidhuber, 1997] policy. Without the LSTM, agents hardly make progress for some mazes, especially if the maze contains two or three goal locations. For this reason, we first generate data for more than 100 mazes and select the first 100 seeds, where the average reward at the end of training is > 0.25. This results in a set of seeds in $[0, 120]$ Otherwise, we use standard hyperparameter settings as provided in `stable-baselines3`.

Source Algorithm performance. We show the average learning curves over all 100 task-specific PPO agents in Figure 11. On average, the agents receive a reward of ≈ 1 over all mazes. This average include environments with one, two or three goals. We provide further dataset statistics for MazeRunner with the corresponding dataset release.

B.3 Meta-World

The Meta-World benchmark [Yu et al., 2020a] consists of 50 challenging robotics tasks, such as opening/closing a window, using a hammer, or pressing buttons. All tasks in Meta-World use a Sawyer robotic arm simulated using the MuJoCo physics engine [Todorov et al., 2012]. The observations and actions are 39-dimensional and 6-dimensional continuous vectors, respectively. As all tasks share the robotic arm, the state, and action spaces remain constant across tasks. All actions are in range $[-1, 1]$. The reward functions are dense and based on distances to the goal locations (exact reward-definitions are provided in Yu et al. [2020a]). Similar to Wolczyk et al. [2021] and Schmied et al. [2024], we limit the episode lengths to 200 interactions. We follow Yu et al. [2020a] and split the 50 Meta-World tasks into 45 training tasks (ML45) and 5 evaluation tasks (ML5). During evaluation, we use deterministic environment resets after episodes, i.e., objects and goal positions are reset to

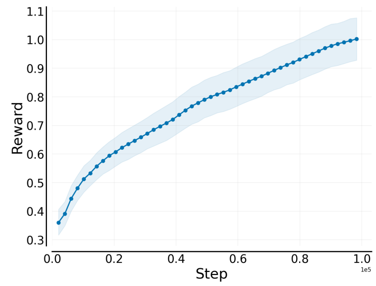


Figure 11: Learning curves for data-collection runs on all 100 mazes on Maze-Runner 15×15 environments with PPO-LSTM as source algorithm. We train for 100K environment steps on each maze and report the mean reward achieved (+ 95% CI).

their original state. Furthermore, we mask-out the goal positions in the state vector, which forces agents to adapt during environment interaction. Agents are given 30 ICL trials during evaluation. The 5 evaluation tasks are:

`bin-picking`, `box-close`, `door-lock`, `door-unlock`, `hand-insert`

Training Dataset. For our Meta-World experiments, we leverage the datasets released by Schmied et al. [2024]. The datasets contain 2M transitions per task, which amounts to 90M transitions across all ML45 training tasks. The data was generated with randomized object and goal positions after every episode.

B.4 DMControl

DMControl contains 30 different robotic tasks with different robot morphologies [Tassa et al., 2018]. Similar to prior work [Hafner et al., 2019; Schmied et al., 2024], we select 16 of these 30 tasks and split them into 11 training (DMC11) and 5 evaluation tasks (DMC5). The DMC11 training tasks are:

`finger-turn_easy`, `fish-upright`, `hopper-stand`, `point_mass-easy`, `walker-stand`, `walker-run`, `ball_in_cup-catch`, `cartpole-swingup`, `cheetah-run`, `finger-spin`, `reacher-easy`

The DMC5 evaluation tasks are:

`cartpole-balance`, `finger-turn_hard`, `pendulum-swingup`, `reacher-hard`, `walker-walk`

States and actions in DMControl are continuous vectors. As DMControl contains different robot morphologies, the state, and action spaces vary considerably across tasks ($3 \leq |\mathcal{S}| \leq 24$, $1 \leq |\mathcal{A}| \leq 6$). All actions in DMControl are bounded by $[-1, 1]$. Episodes last for 1000 environment steps and per time-step a maximum reward of +1 can be achieved, which results in a maximum reward of 1000 per episode. Agents are given 30 ICL trials per task during evaluation, which results in 30K steps for a single evaluation run.

Training Dataset. As for Meta-World, we leverage the datasets released by Schmied et al. [2024]. The datasets contain 1M transitions per task, which amounts to 11M transitions used for training across all DMC11 tasks. We refer to Schmied et al. [2024] for further dataset statistics on DMControl and Meta-World.

B.5 Procgen

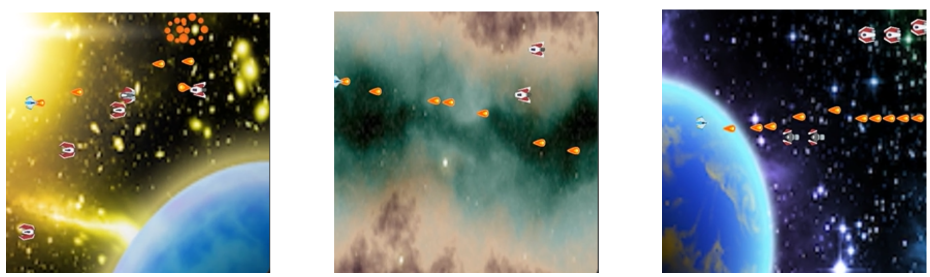
The Procgen benchmark consists of 16 procedurally-generated video games and was designed to test the generalization abilities of RL agents [Cobbe et al., 2020]. Unlike other environments considered in this work, Procgen environments emits $3 \times 64 \times 64$ images as observations. All 16 environments share a common action space of 15 discrete actions. The procedural generation in Procgen is controlled by setting an environment seed. The environments seed randomizes the background and colour of the environment, but retains the same game dynamics. This results in visually diverse observations for the same underlying task, as illustrated in Figure 12 for three seeds on the game `starpilot`. The rewards in Procgen can be dense or sparse depending on the environment.

We follow Raparthy et al. [2023] and use 12 tasks for training and 4 tasks for evaluation, which we refer to as PG12 and PG4, respectively. The PG12 tasks are:

`bigfish`, `bossfight`, `caveflyer`, `chaser`, `coinrun`, `dodgeball`, `fruitbot`, `heist`, `leaper`, `maze`, `miner`, `starpilot`

The PG4 tasks are: `climber`, `ninja`, `plunder`, `jumper`

We exploit the procedural generation of Procgen and evaluate all models in three settings: (1) training tasks - seen seed (PG12-Seen), (2) training tasks - unseen seed (PG12-Unseen), and (3) evaluation tasks - unseen seed (PG4). In particular, the agents observe data from 200 different training seeds. To enable ICL to the same environment, we always keep the same seed during evaluation (seed=1 for PG12-seen, seed=200 for PG12-Unseen and PG4). During evaluation, we limit the episode lengths to 400 steps.



(a) `starpilot`, seed=1 (b) `starpilot`, seed=2 (c) `starpilot`, seed=3

Figure 12: Illustration of procedural generation in Procgen `starpilot`. For different seeds, the same environment looks visually considerably different. We train on multi-task dataset of 12 Procgen tasks, with each dataset containing trajectories from 200 environment seeds. To test for ICL, we evaluate on single hold-out seeds.

Training Dataset. We generate datasets by training task-specific PPO agents for 25M timesteps on 200 environment seeds per task in `easy` difficulty, as proposed in by Cobbe et al. [2020]. We train PPO using the same hyperparameter settings as Cobbe et al. [2020], using a learning rate of $5e^{-4}$, batch size 2048, number of update epochs of 3, entropy coefficient of 0.01, GAE $\lambda = 0.95$, and with reward normalization. We use 256 timesteps per rollout over 64 parallel environments, which results in 16384 environment steps per rollout in total. Furthermore, we found it useful to decrease the discount factor to 0.99.

As in previous experiments, we record the entire replay buffer and consequently, the datasets contain mixed-quality behaviour. We subsample the 25M transitions per task, by storing only the observations of the first 5 parallel environments, which results in approximately 2M transitions per task. To ensure disk-space efficiency, all trajectories are stored in separate `hdf5` files in the lowest compression level files, with all image-observations encoded in `unit8`. Consequently, the datasets for all 16 tasks (32M transitions) take up only 70GB of disk space, and their `hdf5` format enables targeted reading from disk, without loading an entire trajectory into RAM. We release two versions of our datasets: a smaller one containing 2M transitions per task as used in our experiments, and a larger one containing 20M transitions per task.

Source Algorithm performance. We show the individual learning curves for all tasks in Figure 13, and the aggregate statistics over all 16 datasets in Table 1.

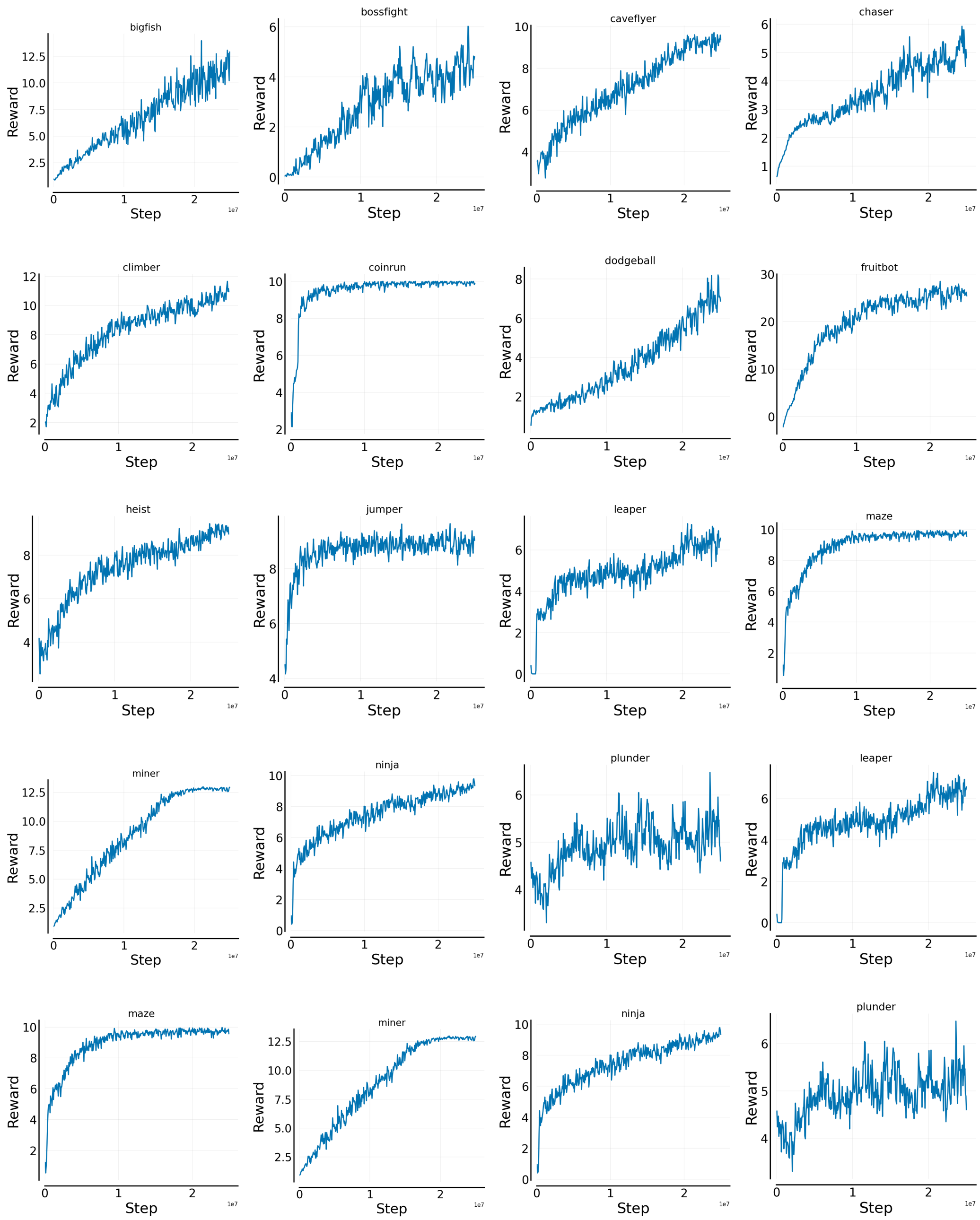


Figure 13: Learning curves for data-collection runs on all 16 Procgen environments with PPO as source algorithm. We train for 25M environment steps on each task in `easy` mode.

Table 1: Dataset Statistics for all 16 Procgen tasks.

Task	**# of Trajectories**	**Mean Length**	**Mean Return**
`bigfish`	8834	221 ± 184	5.9 ± 9.1
`bossfight`	12103	161 ± 200	2.2 ± 4.3
`caveflyer`	16466	119 ± 202	7.6 ± 4.4
`chaser`	9182	213 ± 72	3.4 ± 3.2
`climber`	11392	171 ± 248	9.2 ± 5.2
`coinrun`	38236	51 ± 49	9.7 ± 1.8
`dodgeball`	13089	149 ± 214	3.2 ± 4.2
`fruitbot`	6966	280 ± 152	17.0 ± 14.3
`heist`	8090	241 ± 395	8.0 ± 4.0
`jumper`	45621	43 ± 143	8.7 ± 3.3
`leaper`	28383	69 ± 84	4.9 ± 5.0
`maze`	48867	40 ± 112	9.5 ± 2.3
`miner`	26897	73 ± 182	11.7 ± 3.5
`ninja`	24268	80 ± 136	7.8 ± 4.2
`plunder`	6179	316 ± 106	4.9 ± 3.2
`starpilot`	9490	206 ± 137	17.3 ± 16.4
Average	19628	152	8.2

C EXPERIMENTAL & IMPLEMENTATION DETAILS

C.1 GENERAL

Training & Evaluation. We compare RA-DT against DT, AD, and DPT on all environments. On grid-world environments, we train all methods for 100K steps and evaluate after every 25K steps. For Meta-World, DMControl and Procgen, we train for 200K steps and evaluate after every 50K steps. During evaluation, the agent is given 40 interaction episodes for ICL on Dark-Room and Dark Key-Door, and 30 episodes on MazeRunner, Meta-World, DMControl, and Procgen. We use the ICL curves as the primary evaluation mechanism, and report the scores at the last evaluation step (100K or 200K). Following, Agarwal et al. [2021] we report the mean and 95% confidence intervals across tasks and over 3 seeds in all experiments.

Across experiments, we keep most parameters fixed, unless mentioned otherwise. We train with a batch size of 128 on all environments, except for 40 grids, where we use a batch size of 32. We use a constant learning rate of $1e^{-4}$ and 4000 linear warm-up steps followed by a cosine decay to $1e^{-6}$ and train using the AdamW optimizer [Loshchilov & Hutter, 2018]. Furthermore, we employ gradient clipping of 0.25, weight decay of 0.01, and a dropout rate of 0.2 for all methods.

Context Length. On grid-worlds, we use a context length C equivalent to two 2 episodes for AD, DPT and DT. For example, on 40×20 grids, this results in a sequence length of 6400 ($= 1600 * 4$ for state/action/reward/RTG) for the DT and a sequence length of 4800 for AD. On Meta-World, DMControl and Procgen, we reduce the sequence context length to 50 steps for DT. For RA-DT, we use a shorter context length of $C = 50$ transitions across environments, except for 20×20 and 40×20 grids, where we increase the context length to 100. We want to highlight, that the context length for RA-DT applies to both the input context and the retrieved context. The retrieved context contains the past, and future context, as described in Section 3.2.1. Consequently, the effective context length of RA-DT is $C + 2 * C$ and is independent of the episode length.

Network Architecture. For all environments, except for Procgen, we use a GPT2-like network architecture [Radford et al., 2019] with 4 Transformer layers, 8 head and hidden dimension of 512, which results in 16M parameters. On Procgen, we use a larger model with 6 Transformer blocks, 12 heads and hidden dimension of 768. States, actions, rewards and RTGs are embedded using separate embedding layers per modality, as proposed by Chen et al. [2021]. For all modalities and environments, we use standard linear layers to embed the inputs. Procgen is again an exception, where we use the convolutional architecture proposed by Espeholt et al. [2018] and adopted in prior works [Cobbe et al., 2020; Schmidt & Schmied, 2021; Schwarzer et al., 2023]. Processing image-sequences is computationally demanding. Therefore, we first pre-train the vision-encoder using a separate DT and embed all images in the dataset using the learned vision encoder. Therefore, the data-loading is not bottlenecked by loading entire images into memory, but only their compact representations.

Furthermore, we use global positional embeddings. We also experimented with the Transformer++ recipe (RoPE, SwiGLU, RMSNorm), but only observed minimal performance gains for our problem setting. To speed-up training, we use mixed-precision Micikevicius et al. [2017], model compilation as supported in PyTorch [Paszke et al., 2019], and FlashAttention [Dao, 2023].

Implementation. Our implementation of the DT is based on the `transformers` library [Wolf et al., 2020] and `stable-baselines3` [Raffin et al., 2021]. We integrated AD, DPT, and RA-DT on top of this implementation.

Hardware & Training Times. We run all our experiments on a server equipped with 4 A100 GPUs. For most of our experiments, we only use a single A100. Depending on the environment and method used, training times range from one hour (Dark-Room, DT) to 20 hours (DMControl, AD) for a single training run.

C.2 DECISION TRANSFORMER

For Dark-Room and Dark Key-Door, we sample the target return for RTG conditioning before every episode $\mathcal{N}(90, 5)$, $\mathcal{N}(370, 10)$, and $\mathcal{N}(500, 10)$ for grid sizes 10×10, 20×20, and 40×20, respectively. On grid-worlds, we found that sampling the target return performs better than using a fixed target return per grid size. We assume this is, because specifying a particular target return biases the DT towards particular goal locations. For MazeRunner, we use a constant target return of 3. For Meta-World, DMControl, and Procgen, we set the target return the maximum return achieved for a particular task in the training datasets. However, we also found that constant target returns per domain work decently.

C.3 Algorithm Distillation

AD obtains a context trajectory and learns to predict actions of an input trajectory taken K episodes later. Therefore, we tune K per domain. On grid-worlds, we found $K = 100$ to perform the best, similar to Lee et al. [2023]. For MazeRunner and Meta-World, we set $K = 1000$, and for DMControl and Procgen, we set $K = 250$.

C.4 Retrieval-Augmented Decision Transformer

Embedding Model. For the embedding model $g(\cdot)$, we either use a DT pre-trained on the same environment with the same hyperparameters as listed in Section C, or a pre-trained and frozen LM. For the pre-trained LM, we use `bert-base-uncased` from the `transformers` library by default. BERT is an encoder-only LM with 110M parameters, vocabulary size $v = 30522$, and embedding dimension of $d_{\text{LM}} = 768$ [Devlin et al., 2019]. We apply FrozenHopfield with $\beta = 10$ to state, action, reward and RTG tokens (see Equation 2). To achieve this, we one-hot encode all discrete input tokens, such as actions in Dark-Room/MazeRunner/Procgen or states in Dark-Room, and rewards/RTGs in the sequence before applying the FH. For other tokens, such as continuous states/actions as in Meta-World/DMControl, we directly apply the FH. We evaluate other alternatives for the LM in Appendix E.

Constructing queries/keys/values. Regardless of whether g is domain-specific or domain-agnostic, we obtain C embedded tokens after applying g to the input trajectory τ_{in}. Subsequently, we apply mean aggregation over the context length C to obtain the d_r-dimensional query representation. We experimented with aggregating over all tokens or only tokens of a particular modality (state/action/reward/RTG), and found aggregation over states-only to be most effective (see Appendix E.4). As described in Section 3.2.1, we construct the key-value pairs in our retrieval index by embedding all sub-trajectories in the dataset $\mathcal{D}$ using our embedding model g, $\mathcal{K} \times \mathcal{V} = \{(g(\tau_{i,t-C:t}), \tau_{i,t-C:t+C}) \mid 1 \leq i \leq |\mathcal{D}|\}$. To avoid redundancy, in practice we construct H/C key-value pairs for a given trajectory τ with episode length H and sub-sequence length C, instead of constructing the key and values for every step $t \in [1, H]$. Note that the values, we store $\tau_{i,t-C:t+C}$, contain both the sub-trajectory itself ($\tau_{i,t-C:t}$) and its continuation ($\tau_{i,t:t+C}$). Similar to Borgeaud et al. [2022], we found this choice important for high performance in RA-DT, because it allows the model to observe how the trajectory may evolve if it predicts a certain action (given that the retrieved context is similar enough).

Vector Index. We use Faiss [Johnson et al., 2019; Douze et al., 2024] to instantiate our vector index $\mathcal{I}$. This allows us to search our vector index in $\mathcal{O}(\log M)$ time using Hierarchical Navigable Small World (HNSW) graphs. However, in practice we found it faster to use a Flat index on the GPU as provided by Faiss instead of using HNSW, because our retrieval datasets are small enough. We use retrieval both during training and during inference. It is, however, possible to pre-compute the retrieved trajectories for $\mathcal{D}$ prior to the training phase to limit the computational demand of retrieval, as suggested by Borgeaud et al. [2022]. During evaluation, we can retrieve after every environment step or only after every t environment steps. Here, t represents a trade-off between inference time and final performance. We use $t = 1$ for Dark-Room and Dark Key-Door, and $t = 25$ for all other environments (see Appendix E.6 for an ablation on this design choice). For all environments, except for Meta-World and DMControl, we provide a single retrieved sub-trajectory in the agent's context. For Meta-World and DMControl, we found that providing more than one retrieved sub-trajectory benefits the agent's performance. Therefore, for these two environments, we retrieve the top-4 sub-trajectories, order them by return achieved in that trajectory, and provide their concatenation as retrieved context for RA-DT.

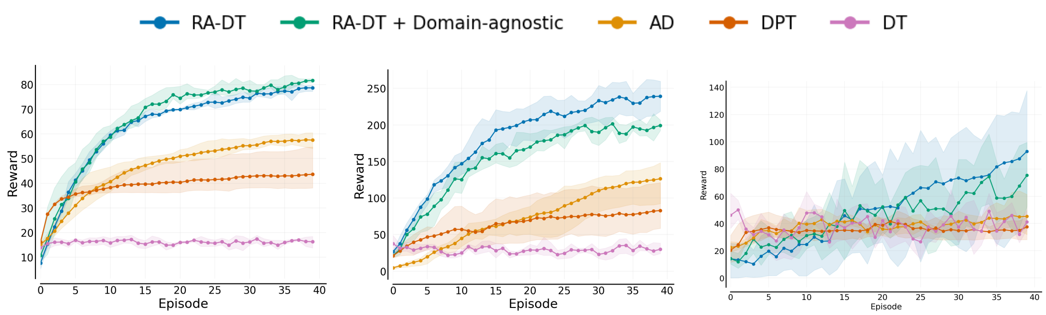
Reweighting. To implement the reweighting mechanism, as described in Section 3.2.3, we first retrieve the top $l \gg k$ experiences and the select the top-k experiences according to their reweighted scores. We set $l = 50$ in all our experiments.

Embedding Retrieved Context. After the most similar trajectories have been retrieved, we embed the state/action/reward/RTG tokens with a separate embedding layers (as is done for the regular input sequence) before incorporating them via the CA layers. We also experimented with sharing/detaching the regular embedding layers, but found it most effective to maintain separate ones. Furthermore, we experimented with an additional Transformer-based encoder for the retrieved sequences, as proposed by Borgeaud et al. [2022], but did not observe substantial performance gains despite increased computational cost.

Retrieval Dataset. For all our experiments, we use the same dataset for retrieval $\mathcal{D}'$ as is used for training $\mathcal{D}$, that is $\mathcal{D}' = \mathcal{D}$. Therefore, we prevent retrieving sub-sequences from the same trajectory as the query.

Retrieval Regularization. We found it advantageous to regularize the k-NN retrieval in RA-DT throughout the training phase. In RL datasets, there is often a substantial overlap between trajectories, leading to many similar sub-trajectories. This poses a significant challenge, as retrieving only similar sub-trajectories encourages the agent to adopt copying behaviour, which renders the DT unable to produce high-reward actions during inference.

One simple strategy to mitigate this issue is **deduplication**, i.e., to discard duplicate experiences before the training phase of RA-DT. To achieve this, we first construct our index as described in Section 3.2. For every key $\mathbf{k} \in \mathbf{K}$, we retrieve the top-$k$ neighbours (excluding experiences from the same episode as $\mathbf{k}$). If the similarity score is above a cosine similarity of 0.98, we discard the experience. This substantially reduces the number of experiences in the index and speeds-up retrieval.



(a) Dark-Room 10×10 (b) Dark-Room 20×20 (c) Dark-Room 40×20

Figure 14: In-context learning performance on **(a) Dark-Room 10×10**, **(b) Dark-Room 20×20**, **(c) Dark-Room 40×20** at end of training (100K steps). We evaluate each agent for 40 episodes on each of the 80 **training tasks** and report mean reward (+ 95% CI) over 3 seeds.

Two other strategies for regularizing retrieval during the training phase, are **similarity cut-off** and **query dropout** [Yasunaga et al., 2023]. Similarity cut-off first retrieves the top $m > l$ experiences, discards the experiences with a similarity score above a threshold (e.g., 0.98), and retains only the remaining experiences l. If used in combination with reweighting, we set $m = 2 * l$. Query dropout randomly drops-out tokens (e.g., 20%) of the embedded sub-trajectory τ_{in}, which leads to more diverse retrieved experiences. We found both strategies effective for RA-DT. We use query dropout of 0.2, similarity cut-off of 0.98, and deduplication by default. Furthermore, for Meta-World and DMControl, we found **query-blending** useful. Query-blending interpolates between then actual query and a randomly selected key from the retrieval index, $\boldsymbol{q}' = \boldsymbol{q} * \alpha_{\text{blend}} + (1 - \alpha)\boldsymbol{q}_{\text{rand}}$. For Meta-World and DMControl we additionally set $\alpha_{\text{blend}} = 0.5$.

On Dark-Room and Dark Key-Door environments, we found it useful to replace retrieved experiences with experiences randomly sampled from the same task, if the query sub-sequence is from the beginning of the episode (i.e., smaller than timestep 10). This is because on these two environments, retrieving appropriate experience can be difficult if the given query sub-sequence is too short.

Finally, we use the same RTG-conditioning strategy as the vanilla DT, as described in Appendix C.2.

Algorithm 2 RA-DT at training time

Input: DT π_θ, embed model g, dataset $\mathcal{D}$, gradient steps N, context len C, batch size B, eval frequency E, loss function $\mathcal{L}$ (cross-entropy or MSE), `evaluate`, batch-wise procedures `retrieve`, `reweight`, and `update`

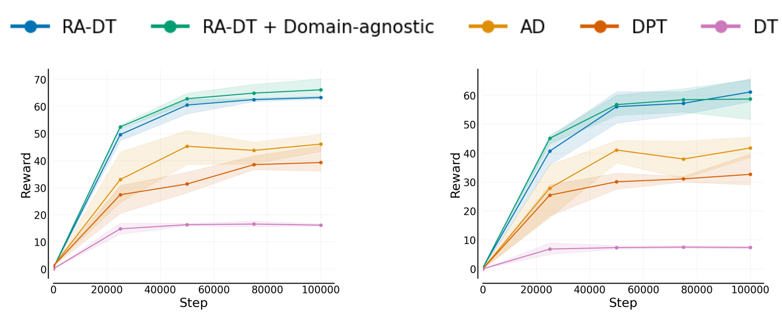
```
I ← ∅                                                   ▷ Initialize retrieval index I
for τ ∈ D do
    I ← I ∪ {(g(τ_{t−C:t}), τ_{t−C:t+C}) | t ∈ range(0, |τ|, C)}   ▷ Add k-v pairs of sub-trjs to I
end for
for i = 1 . . . N do
    b ∼ D where b = {τ_j | 1 ≤ j ≤ B}                   ▷ Sample batch of sub-trjs each of length C
    q = g(b)                                            ▷ Construct queries for all sub-trjs
    R ← retrieve(q, I)                                  ▷ Retrieve top-l sub-trjs, Eq. 3
    S ← reweight(R)                                     ▷ Re-weight top-k sub-trjs, Eq. 4, 5
    a = π_θ(· | b, {τ_ret ∈ S})                         ▷ Predict actions for batch
    π_θ ← update(π_θ, L, a, b)                          ▷ Perform gradient step, see Appendix C.1 for L
    if i % E == 0 then
        evaluate(π_θ, g)                                ▷ Evaluation with ICL, see Algorithm 1
    end if
end for
```

D Additional Results

D.1 Dark-Room

Analogous to the ICL curves on the 20 evaluation tasks in Figure 3, we present ICL curves on the 80 train tasks in Figure 14. In general, we observe a similar learning behaviour on the train tasks as on the evaluation tasks, with slightly higher scores on average. Interestingly, the domain-agnostic variant of RA-DT slightly outperforms its domain-specific counterpart on the training tasks.

In addition, we also show the learning curves on Dark-Room 10×10 over the entire training phase in Figure 15. We evaluate after every 25K updates and observe a steady improvement in the average performances with every evaluation.



(a) 80 Train Goals (b) 20 Eval Goals

Figure 15: Average performances on **Dark-Room 10×10** over the course of training for **(a)** train and **(b)** test tasks. We train each agent for 100K steps and evaluate every 25K steps. Curves are averaged across the 80 train and 20 evaluation tasks, respectively. We report mean reward (+ 95% CI) over 3 seeds.

D.1.1 Attention Map Analysis

We conduct a qualitative analysis on Dark-Room 10×10 to better understand how RA-DT leverages the retrieved context sub-sequences. First, we analyse the attention maps for different Dark-Room 10×10 goal locations.

Table 2: Hyperparameters for RA-DT.

Environment	**Parameter**	**Value**
Default	Gradient steps	100K
	Optimizer	AdamW
	Batch size	128
	Lr schedule	Linear warm-up + Cosine
	Warm-up steps	4000
	Learning rate	1e-4 $\rightarrow$ 1e-6
	Weight decay	0.01
	Gradient clipping	0.25
	Dropout	0.2
	Context Length	50 timesteps
	Top-k before re-weighting	50
	Top-k after re-weighting	1
	Eval steps between retrievals	1
	Query sequence aggregation	mean
	Query sequence tokens	state
	Query dropout	0.2
	Re-weight α	1
	Train re-weighting	task
	Eval re-weighting	return
	Similarity cut-off	0.98
	Deduplicate	True
	Min len for retrieval (only for Dark)	10
	Domain-agnostic LM	`bert-base-uncased`
	Domain-agnostic LM hidden dim	768
	FrozenHopfield β	10
Dark Room/Key-Door 20×20	Context length	100
Dark Room/Key-Door 40×20	Context length	100
	Batch size	32
MazeRunner	Eval steps between retrievals	25
Meta-World/DMControl	Gradient steps	200K
	Eval steps between retrievals	25
	Top-k after re-weighting	4
	Query blending	0.5
Procgen	Gradient steps	200K
	Eval steps between retrievals	25

What happens if an optimal trajectory is retrieved in context? In Figure 16, we showcase this example. The goal location is located at grid cell (4,6). The attention maps exhibit high attention scores for the state and the RTG at the end of the retrieved trajectory. We also observe high attention scores for the state similar to the current state and the action selected in that state. The agent initially imitates the actions in the context trajectory, but deviates further into the episode. Once the agent reaches the goal state, the attention scores for states and RTGs at the end of the trajectory reduce considerably, because the agent need not pay attention to the retrieved context any more.

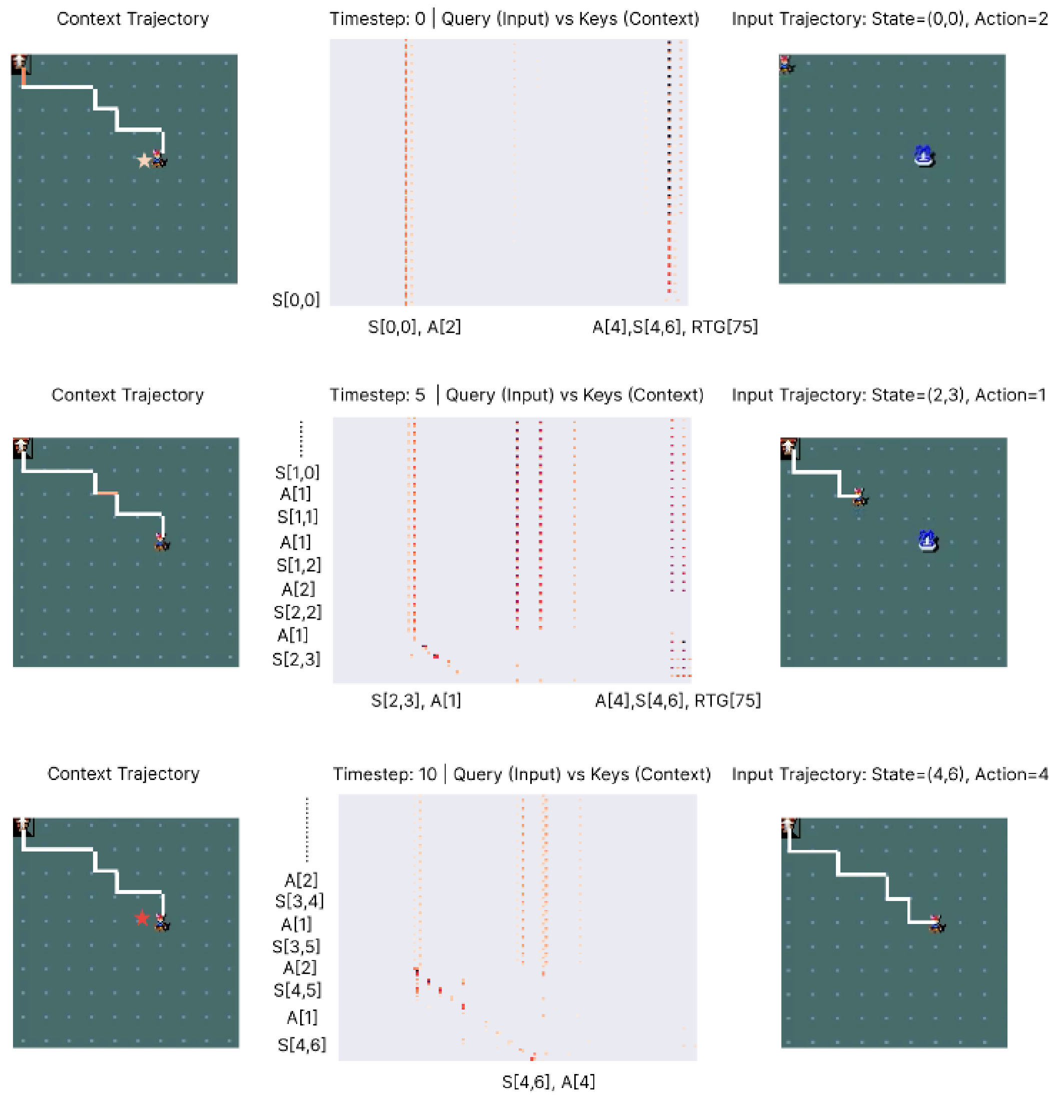


Figure 16: Attention map analysis for an **optimal context-trajectory** on Dark-Room 10×10. We plot the retrieved context trajectory (left), the corresponding attention map, and actual agent state (right), across timesteps (1, 5, 10). Queries (input trajectory) are on the y-axis and keys (context trajectory) on the x-axis. We highlight the sub-sequence in the context trajectory with the highest attention score (left). To improve readability, we mask-out attention scores below a certain threshold, and only provide labels for token that exhibit the highest attention scores. The agent imitates the context trajectory and successfully finds the goal.

What happens if a suboptimal trajectory is retrieved in Context? Similarly, we show the corresponding example in Figure 17. The goal location is again in grid cell (4,6). The retrieved context trajectory reaches the final state (9,5). Similar to Figure 16, the attention maps exhibit high attention scores for the last state and RTG for that state, as well as for a state at a similar timestep. Previously, RA-DT imitated the action, but in this situation the agent picks a different route, as the context trajectory does not lead to a successful outcome.

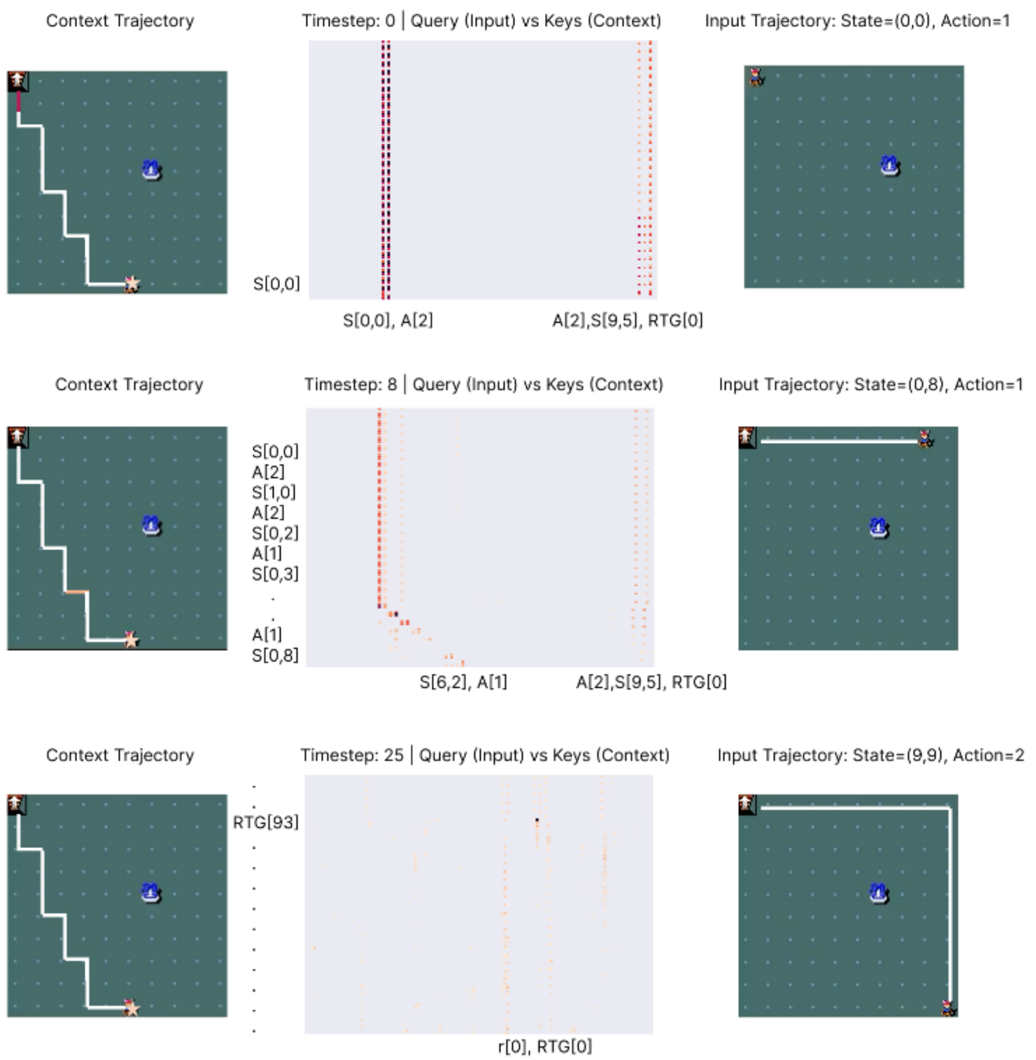


Figure 17: Attention map analysis for a **suboptimal context-trajectory** on Dark-Room 10×10. The agent selects a different route than present in the suboptimal context trajectory and explores the environment.

This analysis suggests, that RA-DT can develop capabilities to either imitate a given positive experience or to behave differently than a given negative experience.

D.1.2 Exploration Analysis

State Visitations. In Section D.1.1, we found that RA-DT learned to either copy or avoid behaviours given positive or negative context trajectories. Therefore, we further analyse the exploration behaviour of RA-DT by visualizing the state-visitation frequencies on Dark-Room 10×10 across the 40 ICL trials for three different goal locations: $(5, 8)$, $(5, 1)$, and $(4, 6)$ (see Figure 18). The agent visits nearly all states at least once at test time, as visualized in Figure 18 (a) and (b). Once the agent finds the goal location, it starts to imitate and stops exploring, as illustrated in Figure 18 (c).

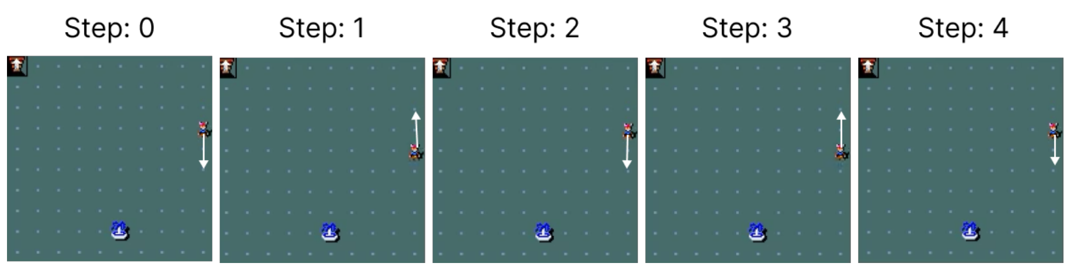
Delusions in RA-DT. Furthermore, we find that in some unsuccessful trials, the agent repeatedly performs the same suboptimal action sequences. Ortega et al. [2021] refer to such behaviour as

(a) Goal Location: (5,8) (b) Goal Location: (5,1) (c) Goal Location (4,6)

Figure 18: We count the state visitations on Dark-Room 10×10 over all ICL trials for three different goal locations: $(5, 8)$, $(5, 1)$, and $(4, 6)$. The total number of states is 100. The agent attempts to visit all states at least once. Once the agent finds the goal, it starts exploiting (e.g., goal location $(5, 1)$).

delusions. In Figure 19, we illustrate two examples in which the agent suffers from delusions and does not recover until the end of the episode.

(a) $(0, 2) \rightarrow (0, 4)$



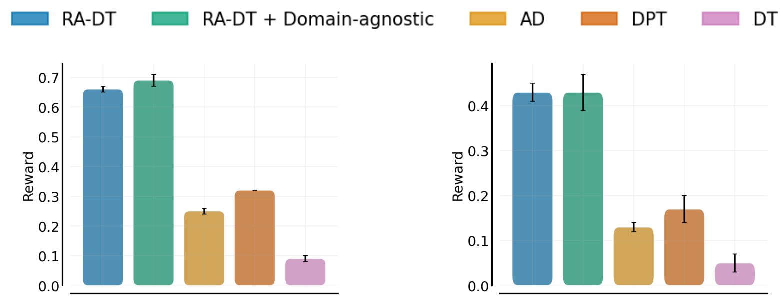
(b) $(3, 9) \rightarrow (4, 9)$

Figure 19: Illustrations of delusions in RA-DT on Dark-Room 10×10. In **(a)**, the agent navigates from state (0, 2) to (0, 4) and returns to (0, 2). In **(b)**, the agent The agent goes from state (3, 9) and (4, 9) and back. In both examples, the agent repeats the unsuccessful action sequence.

D.2 Maze-Runner

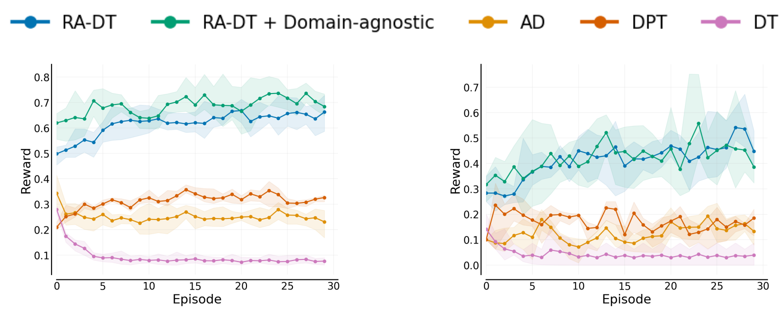
In Figures 20 and 21, we report the average performances at the end of the training (100K) for both the 100 train and 20 evaluation mazes, as well as the corresponding ICL curves, respectively.

While RA-DT outperforms competitors, we observe a considerable performance gap between train mazes and test mazes (0.65 vs. 0.4 reward, see Figure 20). This indicates that RA-DT struggles to solve difficult, unseen mazes. We believe that this gap is an artifact of the small pre-training distribution of 100 mazes, and be closed by increasing the number of pre-training mazes. Furthermore, increasing the number of ICL trials may also enhance the performance.



(a) 100 Train Mazes (b) 20 Test Mazes

Figure 20: Average performance on **(a)** 100 train and **(b)** 20 test mazes at end of training (100K steps). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.



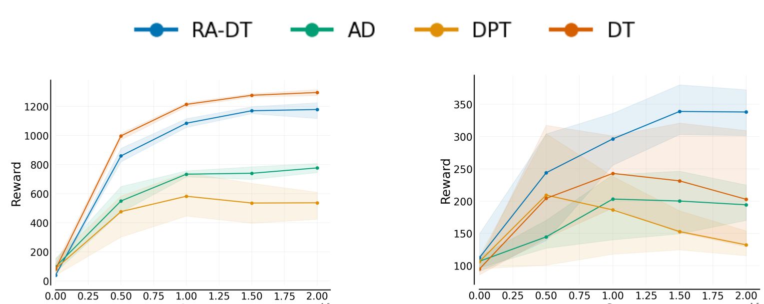
(a) 100 Train Mazes (b) 20 Test Mazes

Figure 21: ICL on **(a)** 100 train and **(b)** 20 test mazes at end of training (100K steps). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.

D.3 Meta-World

In Figures 22 and 23, we show the training curves across the entire training period (200K steps), and the corresponding ICL curves at the end of training for both ML45 and ML5.

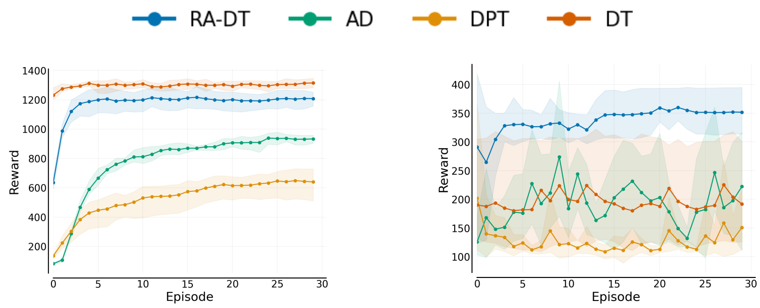
Generally, we observe that RA-DT outperforms competitors on the evaluation tasks in terms of average performance. However, on training task, the average performance of RA-DT is lower than of the vanilla DT. AD and DPT lack behind both methods. One potential reason is the RTG conditioning, which biases DT and RA-DT towards higher quality behaviour.



(a) ML45 (b) MT5

Figure 22: Learning curves on **(a)** ML45 and **(b)** MT5 over the full training period (200K). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.

Nevertheless, we do not observe improved ICL performance of RA-DT on evaluation tasks. While all in-context RL methods exhibit in-context improvement on the training tasks (ML45), neither RA-DT nor other methods show signs of improvement on the evaluation tasks (MT5).



(a) ML45 (b) MT5

Figure 23: ICL performance on **(a)** ML45 and **(b)** MT5 at end of training (200K steps). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.

In addition, we provide the average rewards and data-normalized scores in for the MT5 evaluation tasks in Table 3.

Table 3: Meta-World Evaluation Tasks.

Environment	DT	AD	DPT	RA-DT
		Reward		
bin-picking	62.28 ± 34.37	42.63 ± 17.47	27.52 ± 14.07	14.47 ± 1.79
box-close	70.34 ± 6.72	85.4 ± 14.96	106.79 ± 23.7	110.09 ± 46.69
hand-insert	27.38 ± 3.1	51.82 ± 59.93	13.06 ± 0.15	182.25 ± 99.63
door-lock	229.76 ± 11.4	333.89 ± 161.77	239.2 ± 20.19	219.44 ± 2.51
door-unlock	588.66 ± 454.89	450.71 ± 8.37	249.17 ± 63.38	1163.02 ± 36.42
Average	195.68 ± 97.31	192.89 ± 23.48	127.15 ± 16.97	337.85 ± 34.94
		Data-normalized Scores		
bin-picking	0.24 ± 0.14	0.16 ± 0.07	0.09 ± 0.06	0.04 ± 0.01
box-close	-0.07 ± 0.01	-0.03 ± 0.03	0.01 ± 0.05	0.02 ± 0.1
hand-insert	0.02 ± 0.0	0.04 ± 0.05	0.01 ± 0.0	0.15 ± 0.08
door-lock	0.0 ± 0.01	0.08 ± 0.12	0.01 ± 0.01	-0.0 ± 0.0
door-unlock	0.27 ± 0.31	0.18 ± 0.01	0.04 ± 0.04	0.66 ± 0.02
Average	0.09 ± 0.08	0.08 ± 0.01	0.03 ± 0.03	0.17 ± 0.04

D.4 DMCONTROL

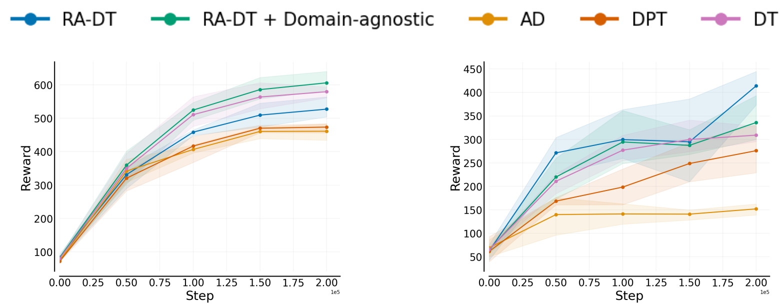
In Figures 24 and 25, we show the training curves across the entire training period (200K steps), and the corresponding ICL curves at the end of training for both DMC11 and DMC5.

Similar to our results on Meta-World, we observe that RA-DT outperforms competitors on average. However, we do not observe in-context improvement on the evaluation tasks.

In addition, we show the average rewards obtained and corresponding data-normalized scores for all DMC5 evaluation tasks in Table 4.

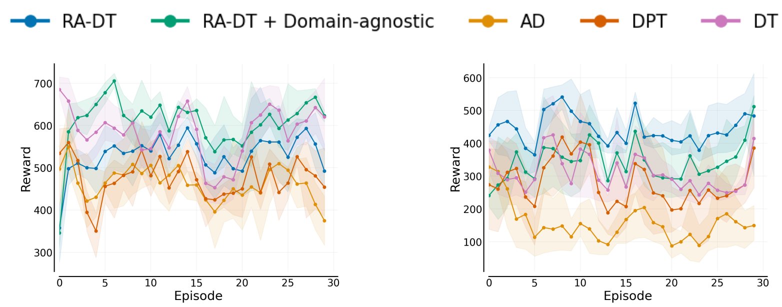
D.5 PROCGEN

In Figures 26 and 27, we show the training curves across the entire training period (200K steps), and the corresponding ICL curves at the end of training for PG12-Seen, PG12-Unseen, and PG4. While



(a) DMC11 (b) DMC5

Figure 24: Average performance on **(a)** DMC11 and **(b)** DMC5 at end of training (200K steps). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.



(a) DMC11 (b) DMC5

Figure 25: ICL performance on **(a)** DMC11 and **(b)** DMC5 at end of training (200K steps). We evaluate each agent for 30 episodes and report mean reward (+ 95% CI) over 3 seeds.

we observe slightly better average performance of RA-DT compared to competitors, we do not find any in-context improvement.

RA-DT constructs bursty sequences.. Building on work by Chan et al. [2022], Raparthy et al. [2023] identified trajectory burstiness as one important property for ICL to emerge on the Procgen benchmark. A given sequence is considered bursty, if it contains at least two trajectories from the same seed (or level). Consequently, the agent obtains relevant information that it can leverage to predict the next action. Therefore, we follow Raparthy et al. [2023] and always provide a trajectory from the same seed in the context of AD and DPT. Indeed, we observed that this improves performance, compared to not taking trajectory burstiness into account. Interestingly, we found that RA-DT retrieves trajectories from the same or similar seeds (seed accuracy of 80%), that is, RA-DT automatically constructs bursty sequences. This intuitively makes sense, as retrieval directly searches for the most relevant experiences (see Section 3.2.3). Therefore, for RA-DT, we do not provide additional information that indicates with which environment seed the trajectory was generated.

E Ablation Studies

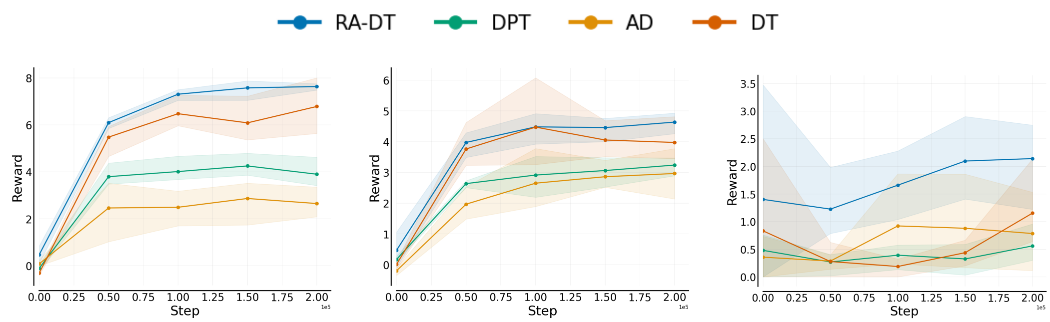
To better understand the effect of learning with retrieval, we presented a number of ablation studies on critical components in RA-DT (Section 4.6). We conduct all ablations on Dark-Room 10×10 and otherwise retain the same experiment design choices, as reported in Section 4.1.

E.1 Retrieval outperforms sampling of experiences

RA-DT is conditioned on sub-trajectories via cross-attention. By default, RA-DT leverages retrieval to search for relevant sub-trajectories for a given input sequence. Instead of retrieval, sub-trajectories can be sampled at random from the external memory. Therefore, we conduct an ablation in which we

Table 4: DMControl Eval Tasks.

Environment	AD	DT	DPT	RA-DT
		Reward		
cartpole-balance	211.96 ± 62.8	946.49 ± 44.91	703.89 ± 263.11	910.1 ± 106.0
finger-turn_hard	199.34 ± 46.0	253.13 ± 43.0	295.2 ± 51.88	336.37 ± 16.51
pendulum-swingup	1.18 ± 2.04	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
reacher-hard	34.22 ± 17.25	167.7 ± 42.86	157.29 ± 94.79	95.4 ± 15.4
walker-walk	326.42 ± 102.52	189.46 ± 10.22	257.11 ± 57.21	877.9 ± 15.2
Average	154.63 ± 12.17	311.36 ± 12.49	282.7 ± 54.62	443.95 ± 25.7
		Data-normalized Score		
cartpole-balance	-0.24 ± 0.11	1.01 ± 0.08	0.6 ± 0.45	0.95 ± 0.18
finger-turn_hard	0.25 ± 0.07	0.33 ± 0.07	0.4 ± 0.08	0.47 ± 0.03
pendulum-swingup	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
reacher-hard	0.03 ± 0.02	0.21 ± 0.06	0.19 ± 0.12	0.11 ± 0.02
walker-walk	0.4 ± 0.14	0.21 ± 0.01	0.31 ± 0.08	1.14 ± 0.02
Average	0.09 ± 0.02	0.35 ± 0.02	0.3 ± 0.09	0.53 ± 0.04



(a) PG12-Seen (b) PG12-Unseen (c) PG4

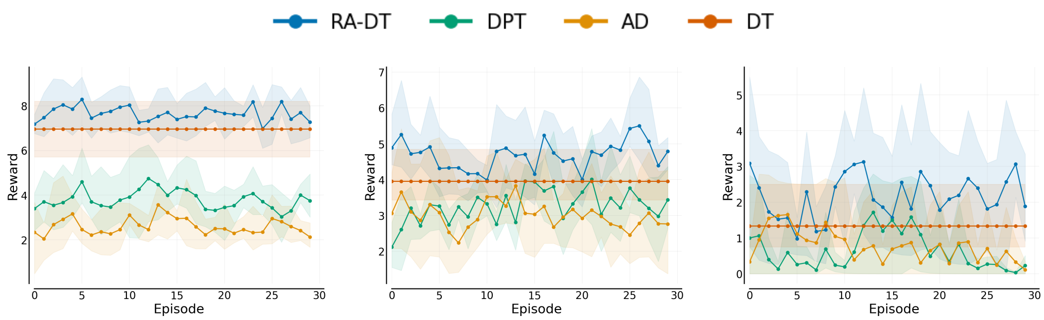
Figure 26: Learning curves on Procgen across **(a)** PG12-Seen, **(b)** PG12-Unseen, and **(c)** PG4 seed over the full training period. We train for 200K steps, evaluate every 50K steps for 30 episodes, and report mean reward (+ 95% CI) over 3 seeds.

swap the retrieval mechanism with random sampling of sub-trajectories during training. This is to investigate the effect of relevance of retrieved sub-trajectories on learning performance. We apply random sampling only during training and use our regular retrieval during inference.

In Figure 6a, we show the ICL curves for training RA-DT with retrieved sub-trajectories, sub-trajectories sampled from the same task as the input sequence, and sub-trajectories sampled uniformly across all tasks. We find that training with retrieval outperforms both sampling variants. Uniform sampling results in poor ICL performance. A reason for this, is that context trajectories from a different goal location, are not relevant for predicting actions in the current sequences. As a result, the model ignores the given context during the training phase, and subsequently is unable to leverage it during inference. In contrast, sampling sub-trajectories from the same task as the input sequence results in better ICL performance, as the model learns to make use of the context trajectories. Nevertheless, using retrieval results in even better ICL performance, as sub-trajectories are not only relevant for the current task, but also similar to the current situation.

E.2 Reweighting Mechanism

Next, we evaluate how our reweighting mechanism affects the ICL abilities of RA-DT. RA-DT reweights a sub-trajectory by its *relevance* and *utility* score (see Section 3.2). During training, we set $s_u(\tau_{\text{ret}}) = 1$, if the τ_{ret} is from the same task as τ_{in}, and 0 otherwise. Instead of reweighting by task ID, alternatives are to reweight a τ_{ret} by its return achieved or by its position in the training dataset.



(a) PG12-Seen (b) PG12-Unseen (c) PG4

Figure 27: ICL performances on Procgen across **(a)** PG12-Seen, **(b)** PG12-Unseen, and **(c)** PG4. We evaluate for 30 episodes, and report mean reward (+ 95% CI) over 3 seeds.

Table 5: Procgen Train Tasks, Train Seeds.

Environment	DT	AD	DPT	RA-DT
		Rewards		
bigfish	4.67 ± 3.51	2.0 ± 0.76	2.41 ± 0.1	5.21 ± 0.25
bossfight	1.0 ± 0.0	0.46 ± 0.55	0.9 ± 0.26	1.31 ± 0.08
caveflyer	3.33 ± 5.77	0.22 ± 0.19	3.0 ± 3.28	9.67 ± 0.0
chaser	1.49 ± 1.05	1.7 ± 0.49	1.64 ± 0.59	2.78 ± 0.46
coinrun	6.67 ± 5.77	5.89 ± 0.69	7.78 ± 1.17	8.33 ± 0.33
dodgeball	7.33 ± 7.57	2.47 ± 0.79	2.8 ± 1.44	8.98 ± 0.87
fruitbot	8.0 ± 2.65	7.66 ± 0.62	7.19 ± 1.09	8.6 ± 0.23
heist	10.0 ± 0.0	0.0 ± 0.0	0.33 ± 0.58	9.11 ± 1.02
leaper	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
maze	10.0 ± 0.0	0.11 ± 0.19	5.56 ± 5.09	8.56 ± 0.69
miner	13.0 ± 0.0	0.94 ± 0.48	1.23 ± 1.15	11.37 ± 0.23
starpilot	18.0 ± 10.54	9.72 ± 4.78	12.9 ± 4.69	17.82 ± 0.72
Avgerage	6.96 ± 1.25	2.6 ± 0.62	3.81 ± 0.68	7.64 ± 0.07
		Human-normalized scores		
bigfish	0.09 ± 0.09	0.03 ± 0.02	0.04 ± 0.0	0.11 ± 0.01
bossfight	0.04 ± 0.0	-0.0 ± 0.04	0.03 ± 0.02	0.06 ± 0.01
caveflyer	-0.02 ± 0.68	-0.39 ± 0.02	-0.06 ± 0.39	0.73 ± 0.0
chaser	0.08 ± 0.08	0.1 ± 0.04	0.09 ± 0.05	0.18 ± 0.04
coinrun	0.33 ± 1.15	0.18 ± 0.14	0.56 ± 0.23	0.67 ± 0.07
dodgeball	0.33 ± 0.43	0.06 ± 0.04	0.07 ± 0.08	0.43 ± 0.05
fruitbot	0.28 ± 0.08	0.27 ± 0.02	0.26 ± 0.03	0.3 ± 0.01
heist	1.0 ± 0.0	-0.54 ± 0.0	-0.49 ± 0.09	0.86 ± 0.16
leaper	-0.43 ± 0.0	-0.43 ± 0.0	-0.43 ± 0.0	-0.43 ± 0.0
maze	1.0 ± 0.0	-0.98 ± 0.04	0.11 ± 1.02	0.71 ± 0.14
miner	1.0 ± 0.0	-0.05 ± 0.04	-0.02 ± 0.1	0.86 ± 0.02
starpilot	0.25 ± 0.17	0.12 ± 0.08	0.17 ± 0.08	0.25 ± 0.01
Average	0.33 ± 0.14	-0.14 ± 0.03	0.03 ± 0.05	0.39 ± 0.0

When reweighting by position, we assign $s_u(\tau_{\text{ret}}) = 1$ if τ_{ret} was generated before τ_{in} by the PPO agent that generated the data. Reweighting by position makes it likely that RA-DT observes the improvement steps in its context.

We find that task-based reweighting is essential for achieving the highest performance scores (see Figure 28). Using no reweighting at all results in a considerable drop in ICL performance. However, using retrieval with no task reweighting still compares favourably to uniform sampling across all

Table 6: Procgen Train Tasks, Evaluation Seeds.

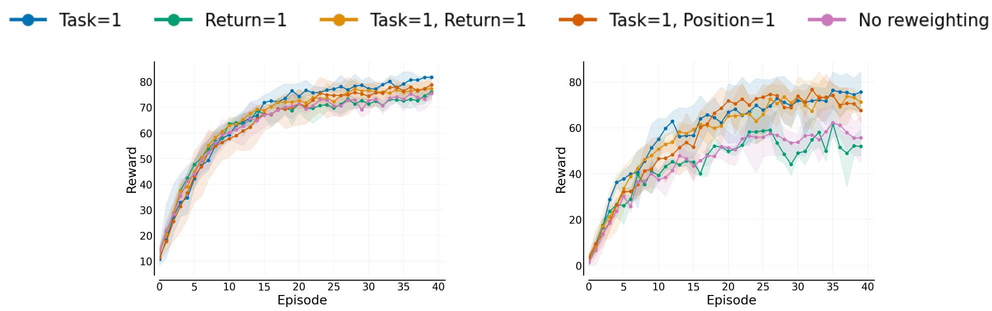
Environment	DT	AD	DPT	RA-DT
		Rewards		
bigfish	0.0 ± 0.0	0.37 ± 0.64	0.04 ± 0.08	0.38 ± 0.3
bossfight	0.33 ± 0.58	0.02 ± 0.02	0.01 ± 0.02	0.02 ± 0.04
caveflyer	6.67 ± 5.77	3.67 ± 2.08	7.67 ± 1.33	9.89 ± 0.19
chaser	2.79 ± 0.65	2.1 ± 0.67	2.75 ± 0.93	5.17 ± 0.58
coinrun	10.0 ± 0.0	9.11 ± 0.84	9.89 ± 0.19	10.0 ± 0.0
dodgeball	0.0 ± 0.0	0.29 ± 0.3	0.0 ± 0.0	0.47 ± 0.41
fruitbot	5.0 ± 4.0	0.63 ± 1.65	1.04 ± 0.83	4.01 ± 1.83
heist	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.11 ± 0.19
leaper	0.0 ± 0.0	0.11 ± 0.19	0.11 ± 0.19	0.22 ± 0.38
maze	6.67 ± 5.77	2.78 ± 4.23	1.67 ± 2.89	8.0 ± 3.46
miner	0.0 ± 0.0	0.58 ± 0.31	0.41 ± 0.07	0.77 ± 0.09
starpilot	16.0 ± 1.0	16.26 ± 5.4	15.81 ± 3.27	17.12 ± 1.58
Average	3.95 ± 0.78	2.99 ± 0.92	3.28 ± 0.26	4.68 ± 0.33
		Human-normalized scores		
bigfish	-0.03 ± 0.0	-0.02 ± 0.02	-0.02 ± 0.0	-0.02 ± 0.01
bossfight	-0.01 ± 0.05	-0.04 ± 0.0	-0.04 ± 0.0	-0.04 ± 0.0
caveflyer	0.37 ± 0.68	0.02 ± 0.24	0.49 ± 0.16	0.75 ± 0.02
chaser	0.18 ± 0.05	0.13 ± 0.05	0.18 ± 0.07	0.37 ± 0.05
coinrun	1.0 ± 0.0	0.82 ± 0.17	0.98 ± 0.04	1.0 ± 0.0
dodgeball	-0.09 ± 0.0	-0.07 ± 0.02	-0.09 ± 0.0	-0.06 ± 0.02
fruitbot	0.19 ± 0.12	0.06 ± 0.05	0.08 ± 0.02	0.16 ± 0.05
heist	-0.54 ± 0.0	-0.54 ± 0.0	-0.54 ± 0.0	-0.52 ± 0.03
leaper	-0.43 ± 0.0	-0.41 ± 0.03	-0.41 ± 0.03	-0.4 ± 0.05
maze	0.33 ± 1.15	-0.44 ± 0.85	-0.67 ± 0.58	0.6 ± 0.69
miner	-0.13 ± 0.0	-0.08 ± 0.03	-0.09 ± 0.01	-0.06 ± 0.01
starpilot	0.22 ± 0.02	0.22 ± 0.09	0.22 ± 0.05	0.24 ± 0.03
Average	0.09 ± 0.1	-0.03 ± 0.1	0.01 ± 0.04	0.17 ± 0.05

Table 7: Procgen Eval Envs.

Environment	DT	AD	DPT	RA-DT
		Reward		
climber	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
ninja	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.89 ± 2.71
plunder	2.0 ± 1.73	0.27 ± 0.13	0.48 ± 0.32	2.39 ± 0.67
jumper	3.33 ± 5.77	2.78 ± 2.83	2.0 ± 1.45	4.33 ± 2.33
Average	1.33 ± 1.01	0.76 ± 0.68	0.62 ± 0.37	2.15 ± 0.85
		Human-normalized Score		
climber	-0.19 ± 0.0	-0.19 ± 0.0	-0.19 ± 0.0	-0.19 ± 0.0
ninja	-0.54 ± 0.0	-0.54 ± 0.0	-0.54 ± 0.0	-0.25 ± 0.42
plunder	-0.1 ± 0.07	-0.17 ± 0.01	-0.16 ± 0.01	-0.08 ± 0.03
jumper	0.05 ± 0.82	-0.03 ± 0.4	-0.14 ± 0.21	0.19 ± 0.33
Average	-0.19 ± 0.19	-0.23 ± 0.1	-0.26 ± 0.05	-0.08 ± 0.12

tasks. This result suggests that retrieval can play an important role in environments without a clear task separation or in scenarios where no task IDs are available.

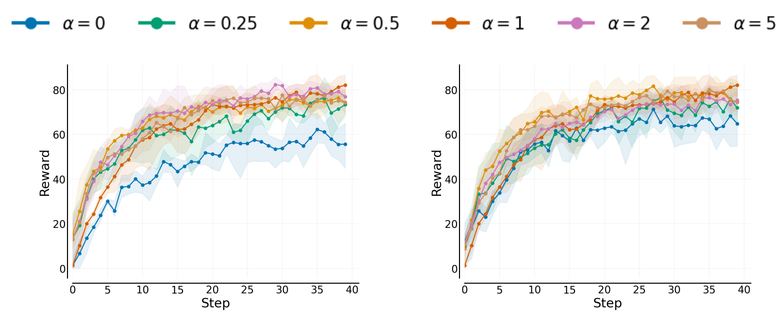
In addition, we conduct a sensitivity analysis on the α parameter used in the re-weighting mechanism that determines how strongly the utility scores influences the final retrieval score. $\alpha = 1$ is used both during training for task-based reweighing and during evaluation for return-based reweighting (see Section 3). In Figure 29, we vary α **(a)** during training, or **(b)** during evaluation, while keeping the



(a) 80 Train Goals (b) 20 Eval Goals

Figure 28: Effect of the **Reweighting Mechanism**. Average performances on Dark-Room 10×10 over the course of training for **(a)** train and **(b)** test tasks.

other fixed. We find that RA-DT perform well for a range of values, but performance declines if no re-weighting is employed ($\alpha = 0$).



(a) Train - Task reweighting (b) Eval - Return reweighting

Figure 29: Sensitivity analysis on α parameter used in re-weighting mechanism of RA-DT on Dark-Room 10×10.

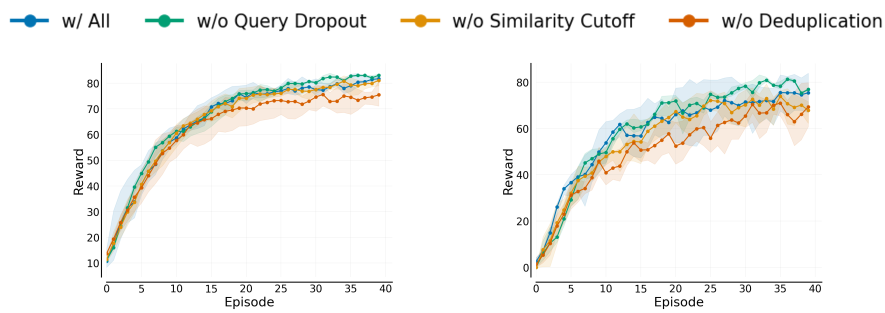
E.3 Retrieval Regularization

Providing the agent with too similar trajectories, can encourage it to adopt copying behaviour instead of generating high-reward actions. To mitigate this, we found it useful to regularize the retrieval using three strategies: deduplication, similarity cut-off, and query dropout. To evaluate their individual impact on ICL performance, we systematically removed each one from RA-DT in Figure 30.

We find that deduplication plays the most significant role in enhancing performance. One reason, why deduplication is effective, is because RL datasets contain many very similar trajectories. Removing overlapping trajectories altogether is therefore beneficial for learning. Notably, deduplication also reduces the index size, thereby speeding-up the search process. The effect of deduplication may vary depending on dataset characteristics, such as state-action coverage [Schweighofer et al., 2022].

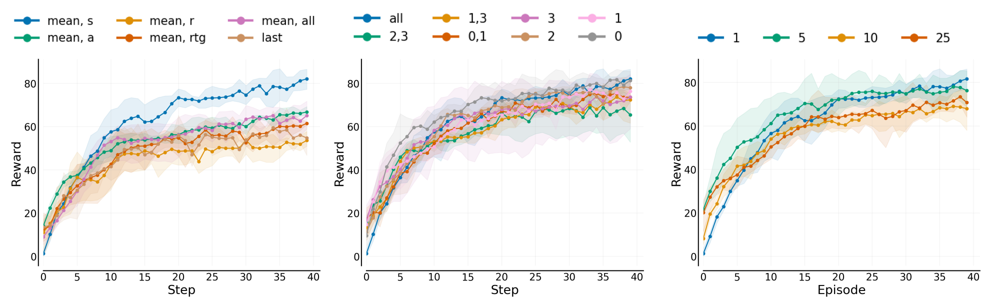
E.4 Query Construction & Sequence Aggregation

In RA-DT, we aggregate the hidden states of an input trajectory using mean aggregation of state tokens over the context length C to obtain the d_r-dimensional query representation. It is, however, possible to use the hidden states of other tokens to construct the query. Therefore, we provide empirical evidence for this design choice in Figure 31a. We compare aggregating states, rewards, actions, returns-to-gos, all tokens, or only using the very last hidden state. Indeed, we find that aggregating state tokens gives the best results.



(a) 80 Train Goals (b) 20 Eval Goals

Figure 30: Effect of **Retrieval Regularization**. Average performances on Dark-Room 10×10 over the course of training for **(a)** train and **(b)** test tasks.



(a) (b) (c)

Figure 31: Ablations on important components of RA-DT conducted on Dark-Room 10×10. In **(a)** we investigate **sequence aggregations** to construct the query for retrieval. By default, we average state-tokens in the sequence (”mean, s”). In **(b)** we vary the **placement of cross-attention layers** in the DT. In **(c)** we vary the number of **steps in-between retrievals** during evaluation. We find that RA-DT delivers robust performance across settings.

E.5 Placement of Cross-Attention Layers

Next, we investigate the effect of the placement of the cross-attention layers in RA-DT. In Figure 31b, we therefore vary the placement of cross-attention layers in RA-DT. By default, we use cross-attention after every self-attention layer. We find that other choice also provide good results. While placing the cross-attention at bottom layers tends to be beneficial, placing them only upper level layers tends to hurt performance.

E.6 Interaction steps between context retrieval

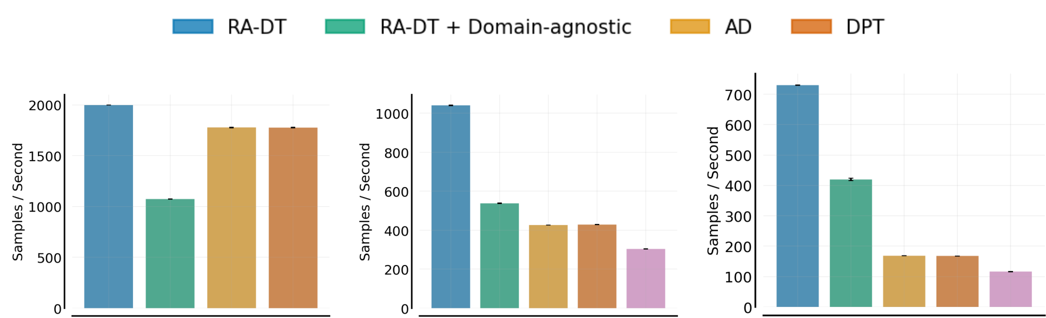
As mentioned in Section C.4, we perform context retrieval after every t environment steps. Here, t represents a trade-off between inference time and final performance. For grid-worlds, we use $t = 1$ by default. To better understand the effect of this design choice, we conduct an ablation in which we vary t (see Figure 31c). Indeed, we find that higher values for t result in a slight decrease in performance, but faster inference.

E.7 Effect of retrieval-augmentation on Training efficiency

Retrieval-augmentation adds computational overhead to the training pipeline due to the cost of embedding the query trajectories and searching for similar experiences in the vector index. Therefore, we study the effect of retrieval-augmentation on the training efficiency of RA-DT. For the purpose of this analysis, we measure training efficiency in terms of the *number of samples processed per second* (higher is better). We run all experiments on an A100 GPU using the same training setup (batch sizes, context lengths) as described in Appendix C.

In Figure 32, we compare domain-specific/agnostic RA-DT to the three considered baselines on Dark-Room across gridsizes 10×10, 20×20, and 40×20. We find that the domain-specific variant of RA-DT attains minor training speed-ups on 10×10 and trains almost $7\times$ faster than baselines on the largest grid. The domain-agnostic variant of RA-DT, in contrast, exhibits slower training times on 10×10, but also trains significantly faster on the largest grid. Note that the differences among the three grid-sizes in the number of samples processed per second of RA-DT stem from the difference in sequence lengths ($C = 50$ for 10×10, $C = 100$ for $20 \times 20/40 \times 20$) and batch sizes ($B = 128$ for $10 \times 10/20 \times 20$, $B = 32$ for 40×20).

The efficiency gains of RA-DT are a direct result of the shorter required sequence lengths. In contrast to the baselines, the computational requirements of RA-DT do not grow with the episode length of the environment. Additional speed-ups can be achieved for RA-DT by pre-computing the retrieved trajectories prior to training similar to Borgeaud et al. [2022]. We also want to highlight that all baselines use FlashAttention to speed-up the training times and to ensure a fair comparison. Consequently, the empirical evidence demonstrates that RA-DT does not only improve the down-stream performance in the environments, but is is also significantly faster to train (up to $7\times$).



(a) Dark-Room 10×10 (b) Dark-Room 20×20 (c) Dark-Room 40×20

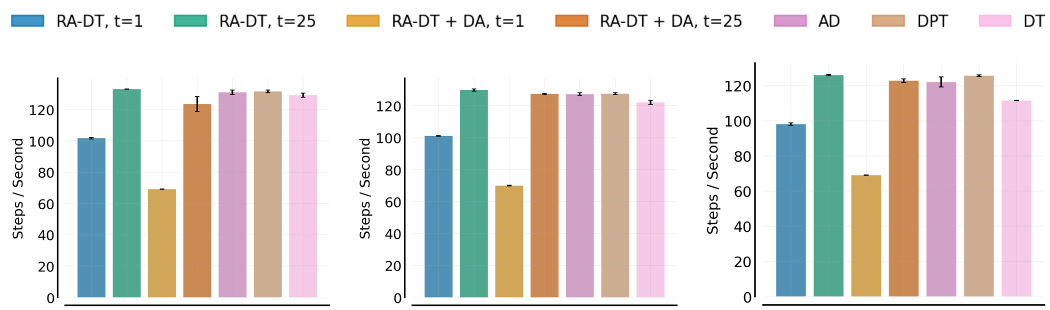
Figure 32: Training efficiency for all considered methods on **(a) Dark-Room 10×10**, **(b) Dark-Room 20×20**, **(c) Dark-Room 40×20**. We measure training efficiency in terms of the number of samples processed per second (higher is better). RA-DT achieves considerably speed-ups, in particular for larger grid-sizes.

E.8 Effect of retrieval-augmentation on Inference efficiency

Retrieval-augmentation also incurs computational overhead during inference. Therefore, we study the effect of retrieval-augmentation on the infernce efficiency of RA-DT, similar to Appendix E.7. We measure inference efficiency in the *number of environment interaction steps performed per second* (higher is better). Note that this metric includes the environment latency. We average the inference efficiency metric across episodes to get a more robust estimate and discard the first episode to exclude compilation times. We conduct our analysis on an A100 GPU and use the same inference setup as described in Appendix C.

In Figure 33, we report the inference efficiency for domain-specific/agnostic RA-DT and the considered baselines on Dark-Room. For RA-DT, we report the inference times with $t \in \{1, 25\}$ where t represents the number of interaction steps between retrievals. In Appendix E.6 we found that increasing t only results in minor performance drops for RA-DT. For $t = 1$ RA-DT exhibits slightly slower inference speeds compared to the baselines. In contrast, for $t = 25$ there is no significant difference in inference speed between RA-DT and the baselines.

Note that the inference speed is roughly the same across grid sizes. This suggests that the inference time is not yet dominated by the quadratic cost of self-attention for $B = 1$ and the sequence lengths we consider in this analysis. To further support this, we run an ablation in which we compare the inference efficiency for all baselines with and without FlashAttention (see Figure 35) on Dark-Room 40×20. Indeed, we observe a significant drop in inference speed when FlashAttention is disabled.



(a) Dark-Room 10×10 (b) Dark-Room 20×20 (c) Dark-Room 40×20

Figure 33: Inference efficiency for all considered methods on **(a) Dark-Room 10×10**, **(b) Dark-Room 20×20**, **(c) Dark-Room 40×20**. We measure inference efficiency in terms of the number of environment interaction steps performend per second (higher is better).

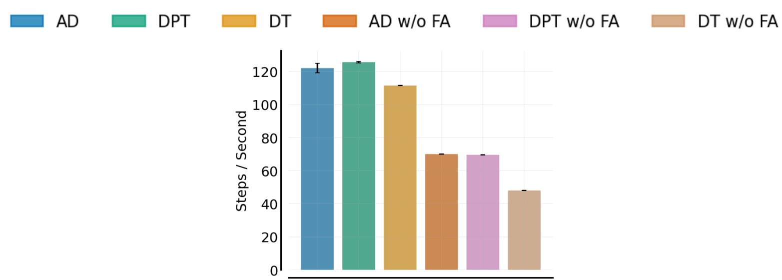


Figure 34: Effect of FlashAttention on inference efficiency of AD, DPT and DT on Dark-Room 40×20. Disabling FlashAttention results in a considerable drop in inference speed.

To conclude this analysis, our findings indicate that while RA-DT is slightly slower when retrieving on every step, it achieves comparable inference speeds to the baselines when retrieving less frequently. Importantly, the retrieval mechanism in RA-DT enables access to the entirety of the experiences collected across all ICL trials. In contrast, the baselines can only access experiences from a limit set of the most recent episodes that are preserved in the context (2 in our experiments). If we were to provide more context episodes to the baselines, the quadratic complexity of self-attention would kick in (similar to Figure 32). The ability of RA-DT to access a much broader set of experiences may be a reason for its enhanced down-stream performance.

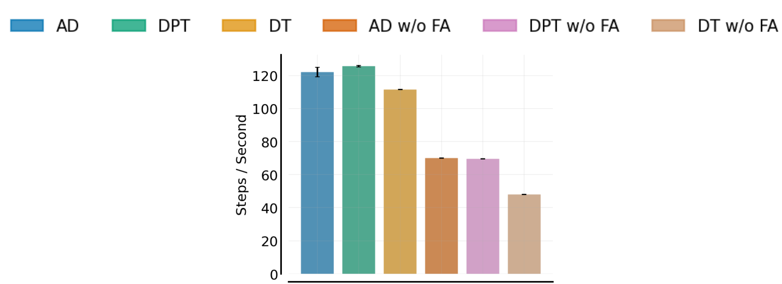
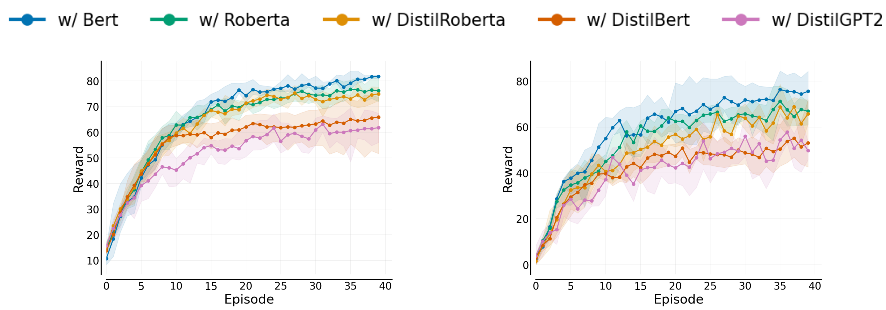


Figure 35: Effect of FlashAttention on inference efficiency of AD, DPT and DT on Dark-Room 40×20. Disabling FlashAttention results in a considerable drop in inference speed.

E.9 PRE-TRAINED LANGUAGE MODEL

We investigate how strongly the ICL performance of RA-DT is influenced by the pre-trained LM used in our domain-agnostic embedding model. In Figure 36, we compare our default choice BERT

[Devlin et al., 2019] against four alternative encoder and decoder backbones, namely RoBERTa [Liu et al., 2019], DistilRoBERTa, DistilBERT [Sanh et al., 2019] and DistilGPT2. We find that RA-DT maintains decent performance across all pre-trained LMs, indicating robust retrieval performance across different LMs. Generally, the non-distilled variants outperform their distilled counterparts. Moreover, this experiment suggests a clear advantage of encoder-only models over the decoder-only LM, DistilGPT2. This suggests that the encoder-only LMs are better able to capture the relations between tokens within the token sequence, which leads to more precise retrieval of sub-trajectories and higher down-stream performance.



(a) 80 Train Goals (b) 20 Eval Goals

Figure 36: Effect of the **Pre-trained LM**. Average performances on Dark-Room 10×10 over the course of training for **(a)** train and **(b)** test tasks.

E.10 Effect of K on Algorithm Distillation

Finally, we investigate the effect of K on the performance of AD. K determines the number of episodes that have passed between the current and the context trajectory, which are provided to AD as the context. Consequently, K specifies the extent of improvement observed between subsequent episodes. By default, we use $K = 100$ for our experiments on Dark-Room 10×10. Therefore, we conduct an ablation study, in which we very K (see Figure 37. We find that too small values for K (e.g., 1 and 10) result in slow ICL behavior. In contrast, too high values for K (e.g., 500) lead to fast initial progress but suboptimal performance in the long term. Only $K = 100$ leads to steady improvement across all interaction episodes. Consequently, AD requires careful tuning of K.

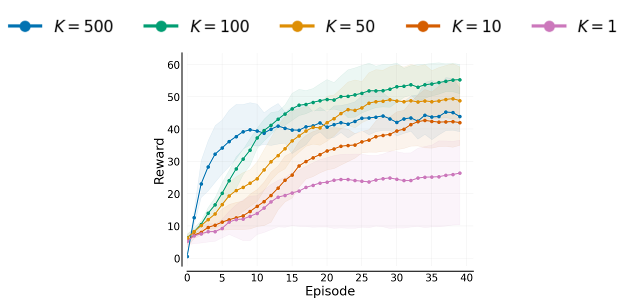


Figure 37: Ablation on the number of episodes K in AD that have passed between "current" trajectory and "context" trajectory on Dark-Room 10×10. K determines how much improvement is observed between episodes. We find that performance increases as K increases, but only up to a certain point ($K = 100$). With $K = 500$, AD improves rapidly in the first few episodes, but then flattens out.

E.11 Convergence of Baselines

In the main experiments on grid-worlds reported in Section 4.1, we found that the baselines AD and DPT only reach sub-optimal performance within the 40 ICL trials. Therefore, we analyse their performance if evaluate for more ICL trials. In Figure 38, we compare the evaluation performance of AD and DPT across 200 ICL trials on the 20 hold-out tasks for Dark-Room 10×10. We find that both method continue to improve towards optimal performance in this environment when given more ICL trials. For this ablation, we found it useful to set $K = 50$ in AD (see Appendix E.10) instead of $K = 100$ as used in our main experiments over 40 ICL trials.

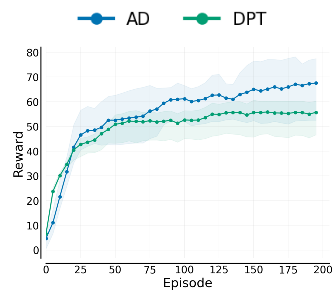


Figure 38: Evaluation of AD and DPT on Dark-Room 10×10 over 200 ICL trials. Both methods continue to improve towards optimal performance on this environment.