# SCALABLE IN-CONTEXT REINFORCEMENT LEARNING WITH RECURRENT ALGORITHM DISTILLATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Algorithm Distillation (AD) has demonstrated the remarkable ability of Transformers to perform in-context reinforcement learning without explicit weight updates. However, capturing long-term learning progress necessitates expansive context windows, which incur prohibitive memory costs and limit scalability in complex, long-horizon tasks. To address this bottleneck, we propose *Recurrent Algorithm Distillation (RAD)*. RAD employs a dual-component architecture: a *Compression Transformer* that distills extended interaction histories into compact latent tokens, and an *AD Transformer* that auto-regressively generates actions using a hybrid context of these compressed memories and recent transitions. By maintaining a fixed-size latent buffer, RAD decouples the effective history length from computational complexity, functionally providing the model with a long-horizon memory. Empirical evaluations across diverse environments demonstrate that RAD matches the asymptotic performance of standard AD with significantly reduced context window sizes, offering a scalable solution for efficient in-context decision-making.

## 1 INTRODUCTION

In-Context Reinforcement Learning (ICLR) enables agents to adapt to new tasks during inference without parameter updates. A prominent example is Algorithm Distillation (AD) (Laskin et al., 2022), which frames RL as a sequence modeling problem. By training on across-episodic learning histories of source RL algorithms, AD distills the "improvement operator" itself, enabling agents to autoregressively predict actions and solve unseen tasks by attending to their own interaction history. Despite its promise, to capture the "improvement operator", AD requires an expansive context window to encompass sufficient transitions, which is expensive to process due to Transformers' quadratic complexity (Vaswani et al., 2017). Pratical implementations therefore truncates older history, which may potentially discard information essential for long-horizon adaptation.

In this work, we argue that this bottleneck arises from a "brute-force" approach to history. In many environments, transitions do not contribute equally to the learning signal; much of the interaction data is redundant, and the essential information can often be represented in a more compact form. We observe that an ideal ICRL agent should be able to maintain a fixed-size representation of the past information while remaining capable of simulating a long horizon. To realize this vision, we propose *Recurrent Algorithm Distillation (RAD)*. As illustrated in Figure 1, RAD replaces the long context window of traditional AD with a dual-transformer architecture designed for recursive context compression. The system consists of: (1) a *Compression Transformer* that distills long interaction sequences into a fixed-length set of latent tokens stored in the long-term memory, and (2) an *AD Transformer* that generates actions by attending to a hybrid context of the latent long-term memories and recent transitions in the working memory. Crucially, as the context window fills, the system recursively recompresses its context to update the long-term memory, enabling the model to preserve information from the distant past while operating under a fixed memory budget.

We evaluate RAD across a diverse suite of RL benchmarks. Our results demonstrate that RAD generally matches or exceeds AD while using substantially shorter contexts, demonstrating that recurrent context compression provides an effective approach to scalable long-horizon ICRL.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

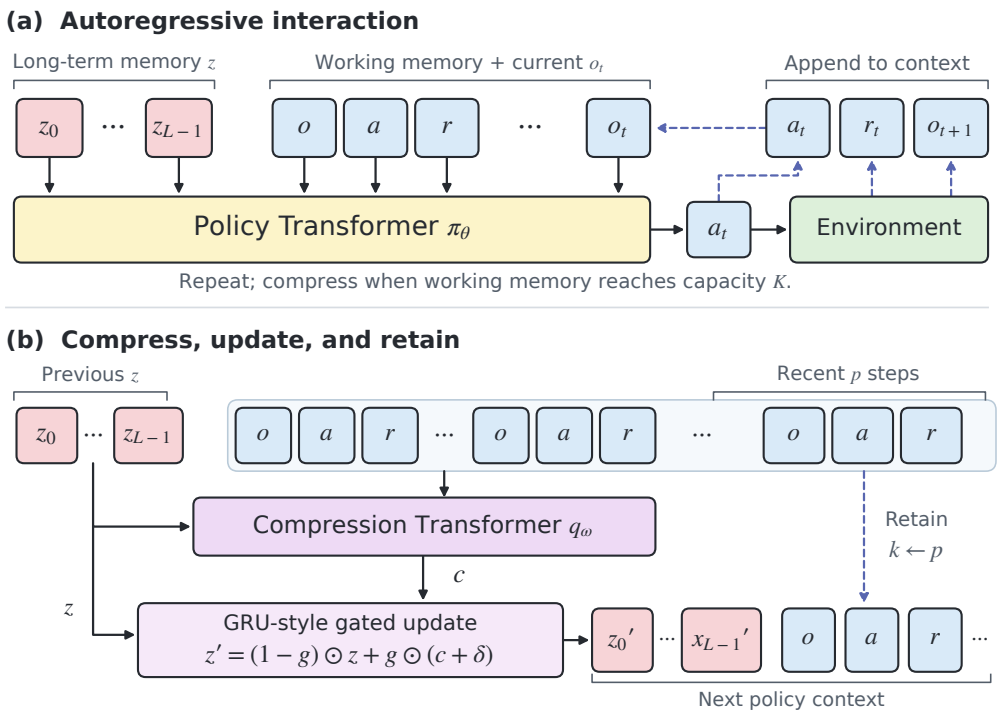



Figure 1: **RAD inference workflow:** During evaluation, the AD Transformer $\pi_\theta$ is able to autoregressively rollout the policy by predicting actions conditioned on the long-term memory $z$ and the working memory $x_{t-k:t}$. When working memory reaches its capacity $K$, the Compression Transformer $q_\omega$ compresses the previous long-term memory (represented as $L$ latent tokens) and the older working-memory context into candidate latent tokens. The update gate $U_\phi$ integrates these candidates with the previous latent state to form the new long-term memory $z'$, while the most recent $p$ steps remain in working memory. At the first compression, since no previous long-term memory exists, $z$ is set directly to $q_\omega(x_{0:K})$, bypassing the update gate.

## 2 Background

**Partially Observable Markov Decision Processes.** We frame the reinforcement learning problem within a *Partially Observable Markov decision process (POMDP)*, defined by the tuple $\mathcal{M} = (\mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R}, \gamma)$. At each time step $t$, the agent selects an action $a_t \in \mathcal{A}$ according to a policy $\pi$, conditioning on the current observation $o_t \in \mathcal{O}$ and the history of past interactions $(o_0, a_0, r_0, \ldots, a_{t-1}, r_{t-1})$. The environment receives $a_t$, transitions to a new configuration according to the transition function $\mathcal{T}$, and provides the agent with a reward $r_t$ and a subsequent observation $o_{t+1}$ based on the reward function $\mathcal{R}$ and the system's dynamics. The agent's objective is maximize the expected return $\mathbb{E}_{\pi,M}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

**Transformers.** Transformers process sequential data using the self-attention mechanism, which allows each token to dynamically aggregate information from other tokens in the sequence. Given query, key, and value representations $Q$, $K$, and $V$, respectively, scaled dot-product attention is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V,$$

where $D$ denotes the model dimension. By directly modeling pairwise dependencies between tokens, self-attention enables Transformers to capture both local and long-range relationships without relying on recurrent computation.

## 3 IN-CONTEXT RL AND ALGORITHM DISTILLATION

We consider the problem of *in-context reinforcement learning (ICRL)*. In this paradigm, an agent performs gradient-free adaptation by utilizing its interaction history—the input context—to internalize environment dynamics and identify optimal behaviors. Unlike standard RL, which relies on gradient-based parameter updates, an ICRL agent leverages its context window to refine its policy during a single forward pass.

*Algorithm Distillation (AD)* (Laskin et al., 2022) formalizes ICRL as a sequential prediction task. The core premise of AD is that the training histories of an RL algorithm inherently encode the logic of policy improvement. By modeling these histories as *long history-conditioned policies* that map past experiences to subsequent actions, AD aims to induce the underlying learning rule directly from data, enabling the agent to solve novel decision-making problems without further weight updates.

We define the *context information* up to time $t$ as:

$$x_t = (o_0, a_0, r_0, \ldots, o_{t-1}, a_{t-1}, r_{t-1}, o_t) \in \mathcal{X} \,.$$

where $\mathcal{X}$ denotes the set of all contexts. An RL algorithm is formalized as a mapping $\phi : \mathcal{X} \to \Delta(\mathcal{A})$, which maps the current context information to a distribution over actions $\Delta(\mathcal{A})$. Given a set of training tasks $\mathcal{M}$ and a task distribution $\rho$, we sample $N$ tasks $\{M_n\}_{n=1}^N \sim \rho$ i.i.d. A dataset $\mathcal{D}$ of learning histories is then collected by running a source algorithm $\phi$ on each task for $T$ steps:

$$\mathcal{D} = \left\{ (o_0^n, a_0^n, r_0^n, \ldots, a_{T-1}^n, r_{T-1}^n, o_T^n) \sim P_\phi(\cdot|M_n) \right\}_{n=1}^N ,$$

where $P_\phi(\cdot|M)$ is the probability distribution over learning sequences induced by executing algorithm $\phi$ on task $M$. AD optimizes a Transformer model $\pi_\theta$ by minimizing the following cross-entropy loss:

$$L_{\mathrm{AD}}(\theta) = -\sum_{n=1}^{N} \sum_{t=0}^{T-1} \log \pi_\theta(a_t^n | x_t^n) \,. \tag{1}$$

This objective enables the transformer model $\pi_\theta$ to internalize the source algorithm's policy improvement patterns in-context. During deployment on a novel task, the model identifies optimal behaviors by auto-regressively sampling actions conditioned on the accumulating interaction history.

**AD as Posterior Sampling** We show that AD implicitly performs posterior sampling. Define the marginal distribution over learning sequences under the task prior $\rho$:

$$P_\phi(x) = \sum_M \rho(M) P_\phi(x|M) \,.$$

The AD loss (1) is a sampled approximation of the cross-entropy between $P_\phi$ and $P_\theta$,

$$\begin{aligned} L_{\mathrm{AD}}(\theta) \approx -\sum_M \rho(M) \sum_x P_\phi(x|M) \log \pi_\theta(x) &= -\sum_x \sum_M \rho(M) P_\phi(x|M) \log \pi_\theta(x) \\ &= -\sum_x P_\phi(x) \log \pi_\theta(x) \,. \end{aligned}$$

Applying the chain rule for cross-entropy, we obtain:

$$L_{\mathrm{AD}}(\theta) \approx \sum_{t=0}^{T} \mathbb{E}_{x_t \sim P_\phi} \left[ -\sum_{a_t} P_\phi(a_t|x_t) \log \pi_\theta(a_t|x_t) \right] .$$

The optimal model thus satisfies $\pi_{\theta^*}(a_t|x_t) = P_\phi(a_t|x_t)$ for all sequences in the support of $P_\phi$. By applying Bayes' rule, we can decompose this optimal policy:

$$\begin{aligned} P_\phi(a_t|x_t) &= \frac{P_\phi(a_t, x_t)}{P_\phi(x_t)} = \frac{\sum_M \rho(M) P_\phi(a_t, x_t|M)}{\sum_{M'} \rho(M') P_\phi(x_t|M')} \\ &= \sum_M \left( \frac{\rho(M) P_\phi(x_t|M)}{\sum_{M'} \rho(M') P_\phi(x_t|M')} \right) P_\phi(a_t|x_t, M) = \sum_M P_\phi(M|x_t) P_\phi(a_t|x_t, M) \,. \end{aligned}$$

This confirms that AD learns a *posterior sampling policy*. Specifically, the model first performs implicit inference to identify the task given the current history and observation, then acts according to the source algorithm $\phi$ for that inferred task:

$$\pi_{\theta^*}(a_t|x_t) = \sum_M P_\phi(M|x_t) P_\phi(a_t|x_t, M)\,. \tag{2}$$

**Scalability Bottleneck of AD** Practical implementations of AD typically employ a fixed-size context window, where older interactions are discarded via a sliding window mechanism once the maximum capacity is reached. However, this method is suboptimal for tasks involving long-range dependencies. Since ICRL relies on the context window to act as a "working memory", the sliding window creates a truncated view of the agent's experience. This limitation forces the agent to rely on a localized and potentially incomplete historical subset, thereby severing the long-term temporal dependencies required for accurate posterior inference (Eq. 2). Consequently, this leads to inconsistent or suboptimal decision-making in complex environments where distal context is critical. This dependency creates a scalability bottleneck, making it difficult to extend AD to complex domains that require extensive historical context.

## 4 Recurrent Algorithm Distillation

This paper explores how to optimize the utilization of long-term historical information in algorithm distillation, addressing the inherent limitations of fixed-length context windows. We introduce *Recurrent Algorithm Distillation (RAD)*, a novel approach featuring a dual-memory architecture. By maintaining a high-fidelity *working memory* alongside a compressed *long-term memory*, RAD allows a Transformer-based policy to condition its in-context decision-making on a significantly extended historical horizon.

### 4.1 Memory Representation

RAD maintains two complementary memory components:

- **Working memory** stores the agent's most recent transitions to capture immediate context. Let $K$ be the working memory's maximum capacity denoted by the number of transitions. At timestep $t$, let $1 \leq k \leq K$ be a lookback window, the contents of the working memory are defined as [1]:

$$x_{t-k:t} = (o_{t-k}, a_{t-k}, r_{t-k}, \ldots, o_{t-1}, a_{t-1}, r_{t-1})\,.$$

- **Long-term Memory** consists of latent representations designed to capture distal dependencies that extend beyond the immediate context. It is structured as a collection of $L$ latent tokens:

$$z_t = (z_{t,0}, z_{t,1}, \ldots, z_{t,L-1})\,,$$

  where $L$ denotes the long-term memory's maximum capacity. The long-term memory is initialized as an empty set, and is updated whenever the working memory reaches its maximum capacity.

The dual-memory architecture enables RAD to compress and preserve critical historical dependencies that would otherwise be lost as the working memory's sliding window advances. We employ a transformer-based compression model based on Context Cascade Compression (C3) (Liu & Qiu, 2025). Given an input sequence of tokens $(y_1, \ldots, y_m)$, the compressor $q_\omega$ produces a compressed sequence of tokens $(y'_1, \ldots, y'_{m'})$ conditioned on $m'$ learnable query tokens with $m' \ll m$. This compression is achieved by jointly training a reconstruction model-also a Transformer-which aims to recover the original sequence $(y_1, \ldots, y_m)$ conditioned on the latent tokens $(y'_1, \ldots, y'_{m'})$

### 4.2 Tokenization

In contrast to Lee et al. (2023) and Son et al. (2025)'s tokenizer which embeds each transition $(o_{t-1}, a_{t-1}, r_{t-1}, o_t)$ into a single embedding vector, at timestep $t$, we treat observation $o_t$, action

[1] We adopt a slight abuse of notation, letting $x_t = x_{0:t}$ represent the complete history when the lookback window $k$ spans the entire trajectory ($k = t$).

$a_t$, reward $r_t$ as distinct tokens, and each is embedded separately into $\mathbb{R}^d$. Each timestep therefore contributes three tokens. To ensure architectural compatibility, the long-term memory is likewise composed of $d$-dimensional latent tokens, allowing the latent state to be directly concatenated with tokenized transitions, forming a unified input sequence for both the policy $\pi_\theta$ and the compression model $q_w$. To preserve the temporal structure, we set $L$ to a multiple of 3, ensuring that compression does not split the $(s_t, a_t, r_t)$ tuple at any timestep $t$.

### 4.3 Inference

RAD autoregressively generates actions conditioned on both memory components. At timestep $t$, given the current long-term memory $z$, the action is sampled as:

$$a_t \sim \pi_\theta(\cdot \mid z, x_{t-k:t}, o_t).$$

RAD then takes an environment step, receiving reward $r_t$ and the next observation $o_{t+1}$. $a_t$ and $r_t$ are appended to the working memory, after which RAD proceeds to predict the next action.

When the working memory reaches its capacity $k = K$, we perform a *compression* update to *distill* critical information from the current context into a new latent state. A compression Transformer $q_\omega$ produces a candidate latent state $c$ from the current context:

$$c = q_\omega(z, x_{t-K:t}) \in \mathbb{R}^{L\times d}.$$

The candidate is integrated with the previous long-term memory using a GRU-style gated update. Denote $v = [z, c]$ as the concatenation along the feature dimension at each latent position. The new latent state is updated as:

$$z \leftarrow (1 - g) \odot z + g \odot (c + \delta),$$

where $g = \sigma(W_g v + b_g)$ and $\delta = \tanh(W_\delta v + b_\delta)$. Particularly, at the first compression event, since the long-term memory is empty, we bypass the gated update and directly set $z \leftarrow q_\omega(x_{0:K})$.

Immediately following the latent state update, the lookback window is set to $p$ where $p << K$, preserving a small window of recent context across memory updates while reserving context for subsequent interactions. This inference procedure is detailed in Algorithm 1.

### 4.4 Training

RAD utilizes the same dataset $\mathcal{D}$ as standard Algorithm Distillation (AD), constructed by collecting the source RL algorithm's learning histories across multiple tasks. We jointly optimize the policy, compressor, and gated memory update through action prediction. Consider a sampled training sequence $x_{t_0:t_0+\ell}$, where $t_0$ is its starting timestep and $\ell = K + nD$. Here, $K$ is the working-memory capacity, $p < K$ is the number of transitions retained after compression, $n$ is the number of compression events, and $D = K - p$ is the number of transitions advanced per compression.

The history incorporated into long-term memory is partitioned into $(n + 1)$ contiguous segments:

$$x_{t_0:t_0+D},\ x_{t_0+D:t_0+2D},\ \ldots,\ x_{t_0+(n-1)D:t_0+nD}, x_{t_0+nD:t_0+\ell}$$

We recursively apply the compressor $q_\omega$ and gated update $U_\phi$ defined in Section 4.3. Starting from empty memory $z_0 = \varnothing$, we obtain latent states $(z_1, \ldots, z_n)$ by

$$z_1 = q_\omega(x_{t_0,t_0+D}), \tag{3}$$

$$z_i = U_\phi(z_{i-1}, q_\omega(z_{i-1}, x_{t_0+(i-1)D:t_0+(i-1)D+K})), \qquad i = 2, \ldots, n \tag{4}$$

The policy is then trained to predict the source learner's actions throughout the final working memory $x_{t_0+nD:t_0+\ell}$. At timestep $t$, it conditions on the final latent state $z_n$, the preceding working-memory history $x_{t_0+nD:t}$, and the current observation $o_t$. The RAD objective is

$$\mathcal{L}_{\mathrm{RAD}}(\theta, \omega, \phi) = -\mathbb{E}_{x_{t_0:t_0+\ell}\sim\mathcal{D}}\left[\frac{1}{K}\sum_{t=t_0+nD}^{t_0+\ell-1} \log \pi_\theta\left(a_t \mid z_n, x_{t_0+nD:t}, o_t\right)\right], \tag{5}$$

where the expectation is over training sequences sampled from the learning histories in $\mathcal{D}$, including their starting timesteps and number of compression events.

To bound the cost of backpropagation through the recurrent memory, we retain gradients through only the most recent $G$ compression events. The training procedure is detailed in Algorithm 2.

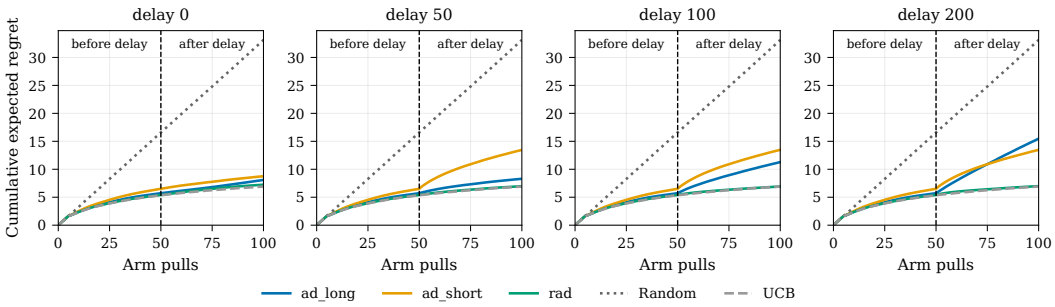



Figure 2: Cumulative regret on Delayed-Bandit. Panels show delays of $D \in \{0, 50, 100, 200\}$ zero-reward distractor steps. Results are averaged over 5 training seeds and 5 evaluate seeds.

# 5 Experiments

## 5.1 Environments

We evaluate the performance of RAD across a diverse set of discrete and continuous environments characterized by varying task complexities and horizon lengths.

**Delayed-Bandit** Delayed-Bandit evaluates the ability to retain useful information across interruptions in a stationary multi-armed bandit task. The base bandit setting follows Lee et al. (2023)'s design, where each arm has a fixed mean sampled independently from $\mathcal{U}[0, 1]$, with rewards drawn from $\mathcal{N}(\mu_a, 0.3^2)$, and a bandit algorithm runs for 100 episodes. However, Delayed-Bandit inserts distractions into the learning histories. The resulting trajectories each consists of 50 arm pulls, followed by $N_{\text{delay}}$ distractor transitions with uniformly sampled actions and zero rewards, and then 50 additional pulls on the same task.

**Grid-World** We consider two discrete grid-world maze environments: **Darkroom** and **Dark Key-to-Door**. In Darkroom, the agent starts at the center of a $9 \times 9$ grid and must locate a hidden goal while observing only its current coordinates. The 81 tasks, defined by distinct goal locations, are partitioned into a 9:1 train-test split. Each episode lasts 20 steps, and the agent receives a reward of 1 upon reaching the goal. Dark Key-to-Door extends this setting by requiring the agent to find a key before reaching the goal, yielding 6561 distinct task configurations and an episode horizon of 50 steps. The agent receives a reward of 1 for finding the key and an additional reward of 1 for reaching the goal, for a maximum episode reward of 2.

**Meta-World** The Meta-World robotic manipulation benchmark suite McLean et al. (2025) is employed to evaluate RAD's performance. We utilize the ML1 tasks, which focus on meta-learning a single task type across 50 different seeds representing varied object and goal locations. These tasks feature horizons of 100 steps.

## 5.2 Dataset Generation

Similar to AD, we first generated a dataset for each environment which consists of training histories of the source RL algorithm solving the training tasks. In this research, we used UCB (Lai & Robbins, 1985) for bandits and PPO (Schulman et al., 2017) for MDPs. Particularly, for Dark Key-to-Door, due to difficulty of exploration in sparse reward settings, we stacked the past 8 observations for the PPO agent as input, but saving only the true observation at the current timestep to ensure compatibility in the training dataset.

## 5.3 Results

The main research question is whether AD can incorporate with a long-term memory buffer and reinforcement learn in context but with a shorter context window. The following are analyzed.

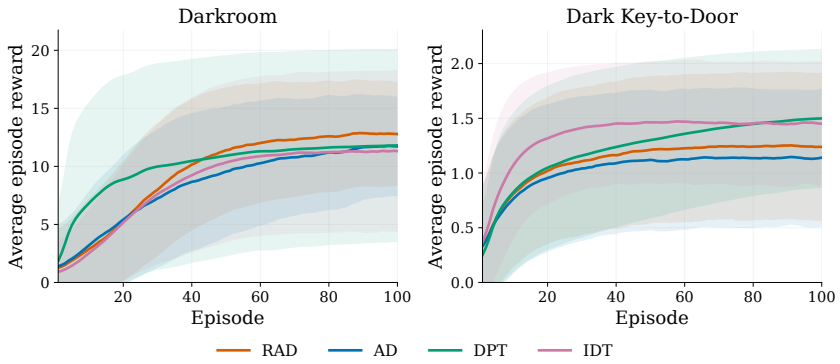



Figure 3: Average episode reward in Darkroom and Dark Key-to-Door, comparing RAD, AD, DPT (Lee et al., 2023), and IDT (Huang et al., 2024). Results are averaged over 3 training seeds and 20 evaluation seeds. RAD outperforms all baseline models in Darkroom, and also gains higher rewards than AD in Dark Key-to-Door. DPT and IDT achieves higher rewards in Dark Key-to-Door, but both requires domain knowledge including optimal actions and desired returns respectively, which generally are not acquirable in regular tasks. On the contrary, though constrained by the high sub-optimality of the source RL algorithm, RAD and AD can autonomously generalize without additional information.

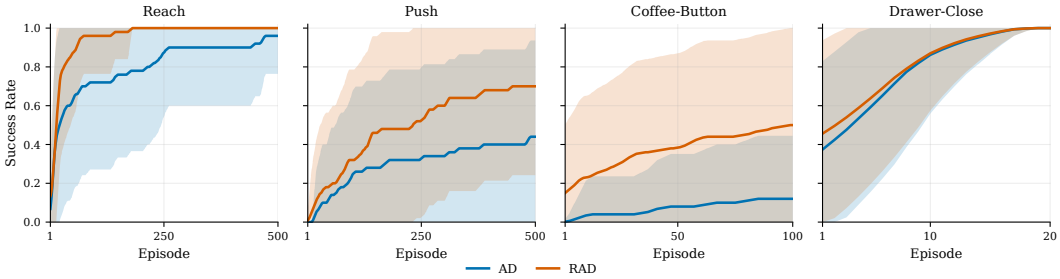



Figure 4: Success rates on Meta-World tasks. Each environment is trained on 50 training tasks, and results are averaged over 50 testing tasks.

**Can RAD effectively perform in-context reinforcement learning on tasks where keeping history is important?** We first collect source data for Delayed-Bandit by running UCB (Lai & Robbins, 1985) for 1000 train tasks, insert $N_{\text{delay}} = 50$ distractor transitions for each trajectory, and then train AD and RAD. We compare two AD variants: $\text{AD}_{\text{short}}$ with the same context window as RAD, and $\text{AD}_{\text{long}}$ with a window exceeding the training trajectory length. We evaluate all models on 100 held-out tasks across varying delay lengths.

Figure 2 shows that all models effectively distills UCB for standard multi-armed bandit tasks with no delay. However, once delay takes place, $\text{AD}_{\text{short}}$ loses pre-delay experience from its sliding window and is forced to re-explore. Both $\text{AD}_{\text{long}}$ and RAD retain useful history through full-context attention and latent memory respectively when $N_{\text{delay}} = 50$, with RAD achieving lower regret. Additionally, RAD generalizes across unseen delays, consistently matching UCB for $N_{\text{delay}} = 100$ and 200, whereas $\text{AD}_{\text{long}}$ degrades substantially. These results suggest that RAD effectively learns latent representations for long-term memories, which is able to filter noises and preserve essential information for action prediction.

**Can RAD explore, assign credit, and generalize in MDP settings?** As shown in Figure 3, RAD outperforms AD on Grid-World tasks despite a potentially shorter active context. Like AD, RAD autonomously generalizes to unseen Grid-World tasks, learning in context to explore, assign credit from sparse rewards, and exploit acquired knowledge without parameter updates. These results suggest that RAD's recursive compression preserves the "improvement operator" learned through distillation.

**Does RAD solve continuous controlling tasks effectively?** Meta-World features a set of continuous robot manipulation tasks with large state and action spaces, emphasizing on generalization across observations and actions. Shown in Figure 4, RAD generally matches and outperforms AD on the tasks like Reach and Push, demonstrating its ability to maintain a useful long-term memory in high-dimensional continuous settings.

**How computationally efficient is RAD?** We profile inference across all experiments and report average FLOPs per action prediction in Table 1. RAD reduced computation for more than 80% in most environment (except for Dark Key-to-Door due to the extreme sparsity in rewards and the hardness of preserving useful information), indicating that RAD provides a scalable solution for efficient in-context decision-making.

Table 1: Inference FLOPs per action for each sequence.

| Environment | AD FLOPs / Action | RAD FLOPs / Action | Compute Ratio ↓ |
|---|---|---|---|
| Bandit | 59.8 M (short) | 43.0 M | **0.72**× |
| | 263.4 M (long) | | **0.16**× |
| Darkroom | 144.8 M | 27.8 M | **0.19**× |
| Dark Key-to-Door | 203.4 M | 85.9 M | **0.42**× |
| Meta-World | 1.84 G | 130.6 M | **0.07**× |

## 6 Ablation Studies

We evaluate and discuss the key design choices in RAD. More details are discussed in Appendix C.

**Effect of the Long-term Memory Update Gate.** We compare the following long-term memory update methods on Darkroom:

- **Replace:** Directly adopt the candidate: $z \leftarrow c$.
- **Residual:** Add the candidate to the old latent state and normalize: $z \leftarrow \mathrm{LayerNorm}(z + c)$.
- **Multiplicative:** Compute a gate from old latents to scale each candidate feature: $z \leftarrow \sigma(W_g z + b_g) \odot c$.
- **GRU:** The method adopted in this work.

Figure 5 shows that GRU gating yields a higher median reward and a narrower interquartile range, supporting its use for recurrent memory updates, possibly due to its role in stabilizing gradient flow during backpropagation through the recurrent memory.

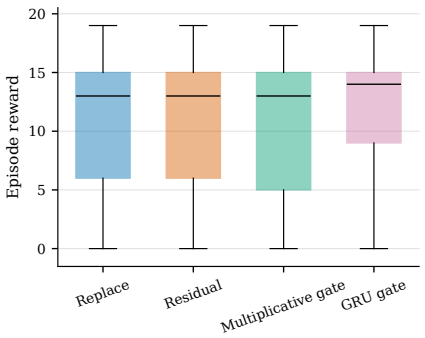



Figure 5: Comparison of different long-term memory update methods.

**Comparison of Different Tokenizers.** We compare our tokenization method with Lee et al. (2023)'s DPT-style tokenizer as introduced in Section 4.2. As illustrated in Figure 6, both tokenization schemes achieved comparable returns for AD and RAD.
However, this work's tokenizer supports more efficient training since its context grows by appending tokens between compression steps, allowing a single Transformer forward pass to supervise multiple observation positions. On the contrary, DPT-style tokenization reconstructs the context at each timestep during inference and therefore can only supervise one observation per input sequence during training.

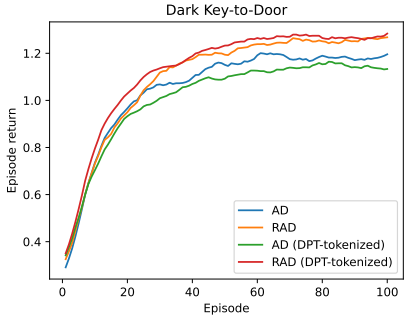



Figure 6: Performance of AD/RAD with different tokenizers on Dark Key-to-Door.

## 7 Related Work

**In-Context Reinforcement Learning.** Transformers have been widely applied to offline RL and sequential decision-making (Chen et al., 2021; Janner et al., 2021; Furuta et al., 2021; Lee et al., 2022; Reed et al., 2022). Beyond learning a fixed policy, in-context RL (ICRL) aims to adapt to new tasks from interaction history without parameter updates. Algorithm Distillation (AD) (Laskin et al., 2022) trains a Transformer on multi-episode RL learning histories to reproduce its learning process in context. Decision-Pretrained Transformer (DPT) (Lee et al., 2023) instead learns to infer optimal actions directly from interaction data. Subsequent work has improved ICRL by injecting noise into training data (Zisman et al., 2023), supporting variable action spaces (Sinii et al., 2023), scaling up high-quality training tasks (Wang et al., 2026), introducing mixture-of-experts architectures (Wu et al., 2026), reweighting action-prediction losses (Dong et al., 2026), and adopting richer prediction objectives, including transition and action-value functions (Son et al., 2025; Mukherjee et al., 2025; Liu et al., 2026). These works primarily improve how in-context policies are learned, whereas RAD focuses on extending the interaction history that can be retained under a fixed memory budget.

**Long-Context Modeling and Memory.** Handling long sequences efficiently has been a long-standing challenge for Transformers. One line of work reduces the computational cost of directly processing long contexts through sparse or linear attention (Beltagy et al., 2020; Zaheer et al., 2020; Katharopoulos et al., 2020; Yang et al., 2025; Team et al., 2025). Another line instead propagates information across bounded segments. Transformer-XL (Dai et al., 2019) caches hidden states from previous segments, while Compressive Transformer (Rae et al., 2019) further compresses evicted activations into a lower-resolution memory. Block-Recurrent Transformer (Hutchins et al., 2022) maintains recurrent state vectors with gated updates across blocks, and Recurrent Memory Transformer (Bulatov et al., 2022) takes a different approach by propagating learned memory tokens between segments, maintaining a fixed-size recurrent representation of past context. More recent approaches such as Infini-attention (Munkhdalai et al., 2024) combine local attention with bounded compressive memory, while Titans (Behrouz et al., 2026) combines attention-based short-term context with persistent neural long-term memory. Retrieval-based approaches such as Memorizing Transformers (Wu et al., 2022) instead maintain an external store of past representations.

Long-context mechanisms have subsequently been applied to reinforcement learning, where agents must reason over extended interaction histories. GTrXL (Parisotto et al., 2020) adapts Transformer-XL-style recurrence to RL, while RATE (Cherepanov et al., 2026b) and ELMUR (Cherepanov et al., 2026a) augment Transformer policies with recurrent or structured external memories for long-horizon decision making. Elastic Decision Transformer (EDT) (Wu et al., 2023) dynamically adjusts the amount of provided trajectory history to facilitate trajectory stitching. For ICRL, Lu et al. (Lu et al., 2023) employ structured state-space models to summarize interaction history in a recurrent latent state, while AMAGO (Grigsby et al., 2024), in contrast, retains a Transformer-based agent and enables end-to-end off-policy RL over long interaction sequences, relying on an explicitly long context as the agent's memory. RA-DT (Schmied et al., 2024) addresses the context bottleneck through retrieval, storing past experiences in an external memory and selecting relevant sub-trajectories for the current decision. RAD takes a complementary approach: rather than retaining or retrieving an expanding set of past experiences, it preserves recent interactions at full fidelity while recursively consolidating older learning history into a fixed-size latent state optimized for in-context action prediction.

## 8 Conclusion

This study has illustrated that Recurrent Algorithm Distillation provides a scalable solution for In-Context Reinforcement Learning. Through a dual-memory architecture with recursive compression, RAD enables efficient long-horizon inference while reducing memory and computational requirements. A key limitation is training stability, as recurrent memory updates require backpropagation through time, increasing optimization and computation difficulty over long horizons. Future work could investigate more stable training objectives, alternative memory-update mechanisms, or approaches that reduce the need for long-range gradient propagation. Overall, RAD provides a promising direction toward long-horizon ICRL agents that retain useful historical information under bounded computational and memory budgets.

### AI USE STATEMENT

In this work, we used generative AI tools for provide feedback on research methodology and experiments, implementing code for the methods, and interpreting results. We have not used generative AI tools for developing research ideas and methods from scratch. Additionally, we used generative AI tools for creating scientific figures, drafting and polishing parts of the paper, as well as gathering and analyzing existing literature. AI-assisted code was reviewed and tested by the authors, and AI-assisted research suggestions were independently evaluated by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide an anonymized implementation of RAD and the experimental code used in this work as supplementary material. The supplementary code includes the model implementation, training and evaluation scripts, environment configurations, and hyperparameters required to reproduce the reported experiments. Additional implementation and experimental details are provided in the main paper and appendices.

## REFERENCES

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 15084–15097. Curran Associates, Inc., 2021.

Egor Cherepanov, Alexey Kovalev, and Aleksandr Panov. Elmur: External layer memory with update/rewrite for long-horizon rl problems. In *International Conference on Learning Representations*, volume 2026, pp. 141814–141844, 2026a.

Egor Cherepanov, Aleksei Staroverov, Alexey Kovalev, and Aleksandr Panov. Recurrent action transformer with memory. In *International Conference on Learning Representations*, volume 2026, pp. 129379–129407, 2026b.

John Co-Reyes, YuXuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *International conference on machine learning*, pp. 1009–1018. PMLR, 2018.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. pp. 2978–2988, 2019.

Juncheng Dong, Moyang Guo, Ethan X Fang, Zhuoran Yang, and Vahid Tarokh. In-context reinforcement learning from suboptimal historical data. *arXiv preprint arXiv:2601.20116*, 2026.

Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta reinforcement learning – identifiability challenges and effective data collection strategies. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 4607–4618. Curran Associates, Inc., 2021.

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.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 06–11 Aug 2017.

Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*, 2021.

Jake Grigsby, Jim Fan, and Yuke Zhu. Amago: Scalable in-context reinforcement learning for adaptive agents. In *International Conference on Learning Representations*, volume 2024, pp. 26919–26952, 2024.

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.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *Advances in neural information processing systems*, 35:33248–33261, 2022.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 1273–1286. Curran Associates, Inc., 2021.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

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 Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 43057–43083. Curran Associates, Inc., 2023.

Kuang-Huei Lee, Ofir Nachum, Mengjiao (Sherry) Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-game decision transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27921–27936. Curran Associates, Inc., 2022.

Fanfan Liu and Haibo Qiu. Context cascade compression: Exploring the upper limits of text compression. *arXiv preprint arXiv:2511.15244*, 2025.

Jinmei Liu, Fuhong Liu, Zhenhong Sun, Jianye Hao, Huaxiong Li, Bo Wang, Daoyi Dong, Chunlin Chen, and Zhi Wang. Scalable in-context q-learning. In *International Conference on Learning Representations*, volume 2026, pp. 97519–97539, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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*, 36:47016–47031, 2023.

Reginald McLean, Evangelos Chatzaroulas, Luc McCutcheon, Frank Röder, Tianhe Yu, Zhanpeng He, K.R. Zentner, Ryan Julian, J K Terry, Isaac Woungang, Nariman Farsad, and Pablo Samuel Castro. Meta-world+: An improved, standardized, RL benchmark. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline meta-reinforcement learning with advantage weighting. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7780–7791. PMLR, 18–24 Jul 2021.

Subhojyoti Mukherjee, Josiah P. Hanna, Qiaomin Xie, and Robert Nowak. Pretraining Decision Transformers with Reward Prediction for In-Context Multi-task Structured Bandit Learning. In *Proceedings of the Reinforcement Learning Conference (RLC)*, August 2025.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

Tianwei Ni, Benjamin Eysenbach, Erfan Seyedsalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive rl. In *International Conference on Learning Representations*, volume 2024, pp. 23555–23569, 2024.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

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.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Gandharv Patil, Aditya Mahajan, and Doina Precup. On learning history-based policies for controlling markov decision processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 3511–3519. PMLR, 2024.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 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.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5331–5340. PMLR, 09–15 Jun 2019.

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

Thomas Schmied, Fabian Paischer, Vihang Patil, Markus Hofmarcher, Razvan Pascanu, and Sepp Hochreiter. Retrieval-augmented decision transformer: External memory for in-context rl. *arXiv preprint arXiv:2410.07071*, 2024.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

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.

Jaehyeon Son, Soochan Lee, and Gunhee Kim. Distilling reinforcement learning algorithms for in-context model-based planning. *arXiv preprint arXiv:2502.19009*, 2025.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Fan Wang, Pengtao Shao, Yiming Zhang, Bo Yu, Shaoshan Liu, Ning Ding, Yang Cao, Yu Kang, and Haifeng Wang. Towards large-scale in-context reinforcement learning by meta-training in randomized worlds. *Advances in Neural Information Processing Systems*, 38:171669–171704, 2026.

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.

Zheng Wang, Cheng Long, and Gao Cong. Trajectory simplification with reinforcement learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 684–695. IEEE, 2021.

Wenhao Wu, Fuhong Liu, Haoru Li, Zican Hu, Daoyi Dong, Chunlin Chen, and Zhi Wang. Mixture-of-experts meets in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 38:24751–24785, 2026.

Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in neural information processing systems*, 36:18532–18550, 2023.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Haoqi Yuan and Zongqing Lu. Robust task representations for offline meta-reinforcement learning via contrastive learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 25747–25759. PMLR, 17–23 Jul 2022.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Ilya Zisman, Vladislav Kurenkov, Alexander Nikulin, Viacheslav Sinii, and Sergey Kolesnikov. Emergence of in-context reinforcement learning from noise distillation. *arXiv preprint arXiv:2312.12275*, 2023.

# A RAD ALGORITHM DETAILS

The following are the pseudocodes for RAD, including inference and training.

**Algorithm 1** RAD Inference

**Require:** AD Transformer $\pi_\theta$, Compression Transformer $q_\omega$, Update Gate $U_\phi$, $K$, $p$

$i \leftarrow 0$, $t \leftarrow 0$, $x \leftarrow ()$, $z_0 \leftarrow \varnothing$
**while** not terminal **do**
  $a_t \sim \pi_\theta(\cdot \mid z_i, x, o_t)$
  $(r_t, o_{t+1}) \leftarrow \textsc{Environment}(a_t)$
  $x \leftarrow x \,\|\, (o_t, a_t, r_t)$
  **if** $|x| = K$ **then**
    **if** $i = 0$ **then**
      $z_{i+1} \leftarrow q_\omega(x)$
    **else**
      $z_{i+1} \leftarrow U_\phi(z_i,\, q_\omega(z_i, x))$
    **end if**
    $x \leftarrow x_{K-p+1:K}$
    $i \leftarrow i + 1$
  **end if**
  $t \leftarrow t + 1$
**end while**

**Algorithm 2** RAD Training

**Require:** Training dataset $\mathcal{D}$, AD Transformer $\pi_\theta$, Compression Transformer $q_\omega$, Update Gate $U_\phi$, Reconstruction Model $d_\psi$, $K$, $p$

$D \leftarrow K - p$
**while** not converged **do** ▷ **Phase 1: Compression Pretraining**
  Sample $x_{t-k:t} \sim \mathcal{D}$
  Update $(\omega, \psi)$ by minimizing

$$\mathcal{L}_{\text{pre}} = \mathbb{E}_{x_{t-k:t} \sim \mathcal{D}} \left[ \| d_\psi(q_\omega(x_{t-k:t})) - y \|_F^2 \right]$$

**end while**
**while** not converged **do** ▷ **Phase 2: Action Distillation**
  Sample $x_{t_0:t_0+\ell} \sim \mathcal{D}$, where $\ell = K + nD$
  **for** $i = 1, \ldots, n$ **do**
    **if** $i = 1$ **then**
      $z_i \leftarrow q_\omega(x_{t_0:t_0+K})$
    **else**
      $z_i \leftarrow U_\phi\big(z_{i-1}, q_\omega\big(z_{i-1}, x_{t_0+(i-1)D:t_0+(i-1)D+K}\big)\big)$
    **end if**
  **end for**
  Compute

$$\mathcal{L}_{\text{RAD}} \leftarrow -\frac{1}{K} \sum_{t=t_0+nD}^{t_0+\ell-1} \log \pi_\theta \left( a_t \mid z_n,\, x_{t_0+nD:t},\, o_t \right)$$

  Update $(\theta, \omega, \phi)$ by minimizing $\mathcal{L}_{\text{RAD}}$
**end while**

# B EXPERIMENT DETAILS

This appendix specifies the architectures, source RL algorithms, and training configurations used for training RAD. We distinguish an environment transition from a Transformer token: each completed transition contributes three tokens, corresponding to its observation, action, and reward. A latent memory token is a learned vector and does not correspond to an individual transition in token length.

### B.1 Model Architectures

**AD Transformer.** The policy backbone is a GPT-2-style decoder-only Transformer that predicts actions from the preceding interaction history. We embed observations, actions, and rewards separately and interleave their embeddings in temporal order, $(o_0, a_0, r_0, o_1, a_1, r_1, \ldots, o_t)$. Learned token-type embeddings distinguish the three modalities, and learned absolute positional embeddings encode their positions within the current context. Grid-World observations are embedded using a lookup table over grid positions. Delayed-Bandit uses an embedding table for the two observation types, genuine interaction and distraction. For Meta-World, a linear layer embeds the 11-dimensional observation used by the sequence model. Discrete actions are one-hot encoded and linearly projected; continuous actions and scalar rewards have separate linear projections.

Each Transformer block consists of multi-head self-attention and a two-layer feed-forward network with GELU activation, residual connections, and pre-layer normalization. A final layer normalization precedes a linear action-prediction head. Action predictions are read from observation-token outputs, with causal attention over the uncompressed history.

**Compression Transformer.** RAD augments the policy backbone with a query-based Compression Transformer. Given embedded history tokens, $m$ learned query tokens are appended to the sequence and the sequence gets forwarded into the Compression Transformer. Each compression layer applies self-attention among the queries, cross-attention from the queries to the input history, and a GELU feed-forward network. Each sublayer uses pre-layer normalization and a residual connection. A final layer normalization is applied to output the candidate updates.

### B.2 Source Reinforcement-Learning Algorithms

We construct the offline datasets from the interaction histories produced while the source algorithms learn each task. These histories preserve the temporal order of experience and the progression of the source learner.

**Upper Confidence Bound (UCB).** UCB balances exploitation of arms with high empirical rewards and exploration of less frequently sampled arms. We use the UCB variant with the exploration bonus used in MBIE-EB (Strehl & Littman, 2008). After pulling each arm once in index order, the algorithm selects

$$a_t = \arg\max_a \left[ \widehat{\mu}_a + \frac{\beta}{\sqrt{N_a}} \right], \qquad \beta = 1, \tag{6}$$

where $N_a$ counts genuine pulls of arm $a$ and $\widehat{\mu}_a$ is its empirical mean reward. Ties are broken in favor of the lowest arm index. During the distractor interval, actions are sampled uniformly and rewards are zero. These transitions remain in the model's input history, while the UCB statistics remain unchanged. Only genuine arm pulls contribute action targets during distillation.

**Proximal Policy Optimization (PPO).** For Grid-World and Meta-World, we use the Stable-Baselines3 (Raffin et al., 2021) implementation of PPO (Schulman et al., 2017). PPO is an on-policy actor–critic method that alternates between collecting rollouts and optimizing a clipped policy surrogate. With probability ratio $\rho_t(\theta) = \pi_\theta(a_t \mid o_t)/\pi_{\theta_{\text{old}}}(a_t \mid o_t)$, its policy objective is

$$\mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_t \Big[ \min\Big( \rho_t(\theta)\widehat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}_t \Big) \Big]. \tag{7}$$

Advantages are estimated using Generalized Advantage Estimation (GAE) (Schulman et al., 2015) and normalized before policy updates. The value function is trained by regression to the estimated returns. The actor and critic each use two hidden layers of 64 units with tanh activations. We use a categorical policy for Grid-World and a diagonal-Gaussian policy for Meta-World. Each source learner collects experience from 100 parallel environment streams for its task. The histories of these streams are retained for sequence-model training.

### B.3 Training Details

**Hardware and optimization.** Experiments were run using eight NVIDIA GeForce RTX 4090 GPUs. Sequence models are implemented in PyTorch. We optimize the sequence models with

AdamW (Loshchilov & Hutter, 2017), using $(\beta_1, \beta_2) = (0.9, 0.99)$, weight decay $0.01$, and gradient clipping at an $\ell_2$ norm of $1.0$. Training uses linear learning-rate warmup followed by cosine decay. Table 4 reports the configured batch sizes and optimization budgets.

**Curriculum.** For Grid-World and Meta-World, policy training gradually increases the allowed number of compression events and shifts sampling toward longer histories. Each training batch uses a single compression-count bucket. Let $c$ denote the number of compressions. We group counts into short ($c = 0$), medium ($c = 1$–$2$), long ($c = 3$–$8$), and very long ($c \geq 9$) categories. Within a category, probability is divided equally among the available counts; probability assigned to an unavailable harder category is added to the largest allowed count. Table 5 gives the stage boundaries, compression limits, and category probabilities.

At each curriculum transition, the learning rate warms up from 10% of its group-specific peak to that peak, then decays cosinusoidally toward 10% over the remainder of the stage. Initial warmup lasts 2,000 steps; later stage warmups last 1,000 steps for Darkroom and Meta-World and 2,000 steps for Dark Key-to-Door. Delayed-Bandit uses no staged compression curriculum and samples genuine-action targets uniformly over the dataset.

### B.4 Hyperparameters

Tables 2–5 provide the architecture, context, source-algorithm, and training settings. DR denotes Darkroom and DKTD denotes Dark Key-to-Door.

Table 2: AD and RAD hyperparameters. RAD uses the same policy architecture as AD. Context windows and capacities are measured in tokens unless otherwise specified (1 transition = 3 tokens).

| Hyperparameter | Delayed-Bandit | DR | DKTD | Meta-World |
|---|---|---|---|---|
| **AD baseline** | | | | |
| Policy model dimension | 64 | 64 | 64 | 64 |
| Policy Transformer layers | 4 | 4 | 4 | 4 |
| Policy attention heads | 4 | 4 | 4 | 8 |
| Policy feed-forward dimension | 256 | 256 | 256 | 256 |
| Attention / residual dropout | 0.1 | 0.1 | 0.1 | 0.1 |
| Policy context window | 150 / 900 | 240 | 300 | 1200 |
| **RAD** | | | | |
| Compression model dimension | 64 | 64 | 64 | 64 |
| Compression Transformer layers | 2 | 3 | 4 | 4 |
| Compression attention heads | 4 | 4 | 4 | 4 |
| Compression feed-forward dimension | 256 | 256 | 256 | 256 |
| Compression dropout | 0.1 | 0.1 | 0.1 | 0.1 |
| Latent memory tokens $m$ | 15 | 15 | 60 | 60 |
| Working memory size (transition) $K$ | 50 | 25 | 52 | 80 |
| Retained recent transitions $p$ | 5 | 5 | 10 | 20 |
| Gradient-bearing compression rounds $G$ | $\infty$ | 5 | 5 | 6 |
| Policy context window | 165 | 90 | 216 | 300 |

Table 3: PPO source-algorithm configuration.

| Hyperparameter | DR | DKTD | Meta-World |
|---|---|---|---|
| Parallel streams per learner | 100 | 100 | 100 |
| Rollout steps per stream | 80 | 50 | 100 |
| Optimization minibatch size | 40 | 100 | 200 |
| Epochs per rollout update | 20 | 10 | 20 |
| Total Timesteps | 100,000 | 100,000 | 1,000,000 |
| Learning rate | $3\times10^{-4}$ | $3\times10^{-4}$ | $3\times10^{-4}$ |
| Discount factor $\gamma$ | 0.99 | 0.99 | 0.99 |
| GAE parameter $\lambda$ | 0.95 | 0.95 | 0.95 |
| Policy clip range $\epsilon$ | 0.2 | 0.2 | 0.2 |
| Value-loss coefficient | 0.5 | 0.5 | 0.5 |
| Entropy coefficient | 0 | 0 | 0 |
| Gradient clip norm | 0.5 | 0.5 | 0.5 |

All three configurations use tanh activations, Adam with its default $(\beta_1, \beta_2) = (0.9, 0.999)$, optimizer epsilon $10^{-5}$, and no weight decay. are inherited from Stable-Baselines3 and PyTorch.

Table 4: AD/RAD training hyperparameters

| Hyperparameter | Delayed-Bandit | DR | DKTD | Meta-World |
|---|---|---|---|---|
| *AD policy training* | | | | |
| Training steps | 50k / 100k[a] | 50k | 50k | 50k |
| Configured batch size | 64 | 512 | 512 | 256 |
| Peak learning rate | $3\times10^{-4}$ | $3\times10^{-4}$ | $3\times10^{-4}$ | $3\times10^{-4}$ |
| Warmup steps | 500 | 1,000 | 1,000 | 5,000 |
| *RAD compression pretraining* | | | | |
| Training steps | 20k | 40k | 50k | 50k |
| Configured batch size | 64 | 512 | 1,024 | 256 |
| Peak learning rate | $10^{-4}$ | $3\times10^{-4}$ | $3\times10^{-4}$ | $3\times10^{-4}$ |
| Warmup steps | 500 | 3,000 | 2,000 | 3,000 |
| *RAD policy training* | | | | |
| Training steps | 100k | 100k | 100k | 100k |
| Configured batch size | 64 | 256 | 1,024 | 128 |
| Policy / compressor / latent LR[b] | 3 / 1 / 3 | 3 / 1 / 3 | 3 / 1 / 3 | 3 / 1 / 3 |
| Initial warmup steps | 500 | 2,000 | 2,000 | 2,000 |
| Later stage warmup steps | — | 1,000 | 2,000 | 1,000 |
| Learning-rate floor / peak | 0 | 0.1 | 0.1 | 0.1 |

[a] The two values correspond to AD-Short / AD-Long.
[b] All three learning rates are in units of $10^{-4}$.

Table 5: RAD curriculum. Probabilities correspond to short, medium, long, and very-long compression-count categories.

| Environment | Start step | Max. $c$ | Short | Medium | Long | Very long |
|---|---|---|---|---|---|---|
| DR | 0 | 1 | 0.60 | 0.35 | 0.05 | 0.00 |
| DR | 30k | 3 | 0.35 | 0.40 | 0.20 | 0.05 |
| DR | 50k | 6 | 0.25 | 0.30 | 0.30 | 0.15 |
| DR | 75k | $\infty$ | 0.25 | 0.25 | 0.25 | 0.25 |
| DKTD | 0 | 2 | 0.30 | 0.55 | 0.15 | 0.00 |
| DKTD | 25k | 4 | 0.20 | 0.45 | 0.30 | 0.05 |
| DKTD | 50k | 6 | 0.15 | 0.35 | 0.35 | 0.15 |
| DKTD | 75k | $\infty$ | 0.15 | 0.30 | 0.30 | 0.25 |
| Meta-World | 0 | 2 | 0.30 | 0.55 | 0.15 | 0.00 |
| Meta-World | 25k | 4 | 0.20 | 0.45 | 0.30 | 0.05 |
| Meta-World | 50k | 6 | 0.15 | 0.35 | 0.35 | 0.15 |
| Meta-World | 75k | $\infty$ | 0.15 | 0.30 | 0.30 | 0.25 |

# C Extended Experiments

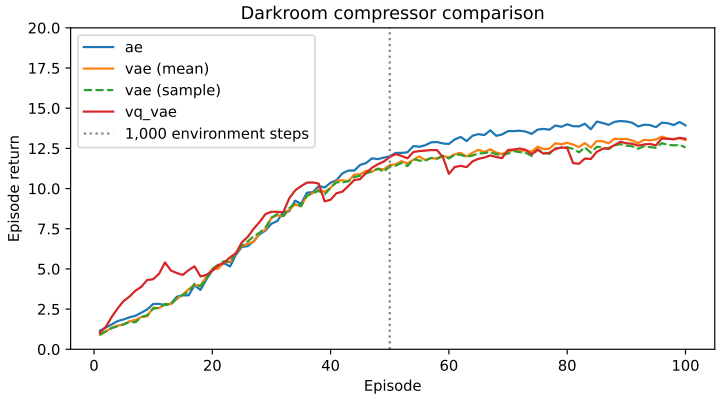



Figure 7: Compressor ablation in Darkroom. Episode returns for RAD with AE, VAE (posterior mean or sampling), and VQ-VAE compressors. AE achieves the highest late-stage returns. The vertical dotted line marks the 1,000-step training-history horizon (50 episodes).

**Comparison of Different Compressors.** We compare three Transformer-based compressors in Darkroom, using the same backbone, latent-token dimensions, and recurrent memory update:

- **AE:** the compressor adopted in this work, featuring a deterministic autoencoder with continuous latent representations.
- **VAE:** a variational autoencoder with a Gaussian latent distribution and KL regularization, evaluated using either posterior means or sampled latents (Kingma & Welling, 2013).
- **VQ-VAE:** a vector-quantized autoencoder that maps compressor outputs to a learned discrete codebook before the recurrent memory update (Van Den Oord et al., 2017).

Shown in Figure 7, all variants improve with accumulated interaction. Its advantage becomes clearer beyond 50 episodes, corresponding to the 1,000-step training-history horizon. VAE performs similarly with posterior means and sampled latents, with a small late-stage advantage for mean-based evaluation. VQ-VAE initially improves faster but exhibits larger fluctuations and ultimately attains returns comparable to VAE. These results favor the deterministic continuous AE for this setting: neither variational regularization nor vector quantization improves downstream performance.

**Effect of the Long-term memory size.** Figure 8 shows the performance of RAD with the same working memory size and different long-term memory size. RAD successfully exhibit in-context adaptation even for a small long-term memory, possibly since preserving information in Darkroom is relatively simple. A long-term memory with 15 latent tokens exhibit the best performance while maintaining a relatively small long-term memory.

**Convergence speed of AD/RAD.** Figure 9 tracks online regret on held-out tasks as a function of the number of training updates; all three variants start from a regret of $\approx$33 at initialization and improve steadily thereafter. AD-short descends fastest early but saturates at a suboptimal plateau, since its short context cannot fully resolve feedback that is 50 steps old; AD-long, whose context spans the entire history, learns more slowly—it still trails AD-short at 20k updates—but does not saturate, crossing below AD-short around 25–30k updates and converging to 6.9. RAD converges the slowest among the methods, achieving the optimal performance at around 60k updates, yet it is more efficient during inference, matching AD-long's performance with significantly smaller context window.

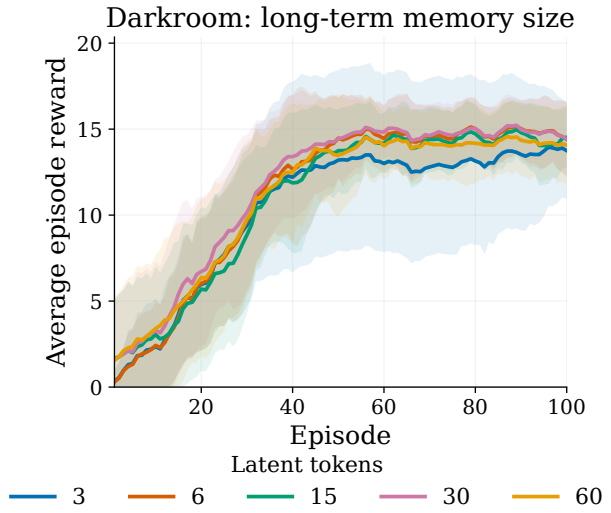



Figure 8: Average rewards of RAD with different long-term memory size on Darkroom

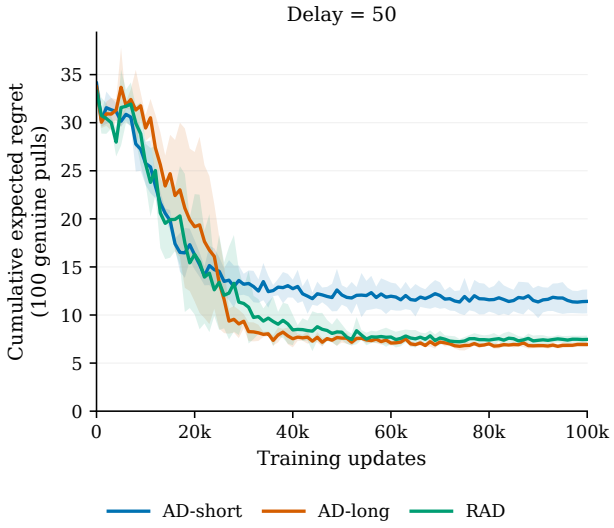



Figure 9: Training convergence of AD-short, AD-long, and RAD under delayed feedback (delay = 50): mean cumulative expected regret over 100 genuine pulls (50 before and 50 after the delay) on 100 fixed held-out tasks, evaluated periodically during distillation, with shading showing one standard deviation across 3 training seeds; RAD starts from compression-pretrained checkpoints (pretraining updates excluded from the horizontal axis), and all methods receive the same distillation update budget.

# D Extended Related Work

**History and Trajectory Representations in RL.** RAD is also related to methods that learn compact representations of trajectories or interaction histories. SeCTAR (Co-Reyes et al., 2018) learns latent trajectory representations for hierarchical RL, while history-based representation methods study how past observations can be summarized into compact states for control (Patil et al., 2024; Ni et al., 2024). HELM (Paischer et al., 2022) similarly constructs compact history representations for partially observable RL. Trajectory simplification has also been formulated as an RL problem (Wang et al., 2021), where redundant trajectory points are removed while preserving geometric information. These methods primarily seek representations sufficient for control, prediction, or trajectory reconstruction. RAD instead targets the learning history of an in-context RL agent: its long-term memory recursively summarizes experience across episodes and is optimized to retain the information needed to continue the adaptation process of the distilled source algorithm.

**Meta Reinforcement Learning.** Meta-reinforcement learning (Meta-RL) aims to learn the underlying structure of RL, enabling rapid adaptation to new tasks. Early deep meta-RL methods primarily focused on online adaptation, where recurrent architectures are trained to implicitly implement RL algorithms through their internal states, enabling fast within-episode learning (Duan et al., 2016; Wang et al., 2016). On the other hand, gradient-based meta-learning approaches seek to learn parameter initializations that can be efficiently adapted to new tasks via a small number of gradient updates (Finn et al., 2017; Nichol et al., 2018).

More recently, offline meta-RL has emerged as an important paradigm by leveraging pre-collected datasets of trajectories to meta-train without additional environment interaction. Existing offline meta-RL methods span multiple classes, including gradient-based adaptation (Mitchell et al., 2021), Bayesian inference over latent task variables (Rakelly et al., 2019; Dorfman et al., 2021), and representation-learning or contrastive approaches for task inference from offline data (Yuan & Lu, 2022).