DUAL-LOOP AGENTS: HUMAN-LEVEL ATARI VIA PRIOR-GUIDED SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

General self-improving agents must turn prior knowledge and limited online experience into effective action. However, existing approaches expose a tension: reward-driven reinforcement learning (RL) may fail to discover long-horizon strategies, whereas foundation-model priors often leave primitive actions unspecified. We introduce *dual-loop agents*, a self-improvement framework unifying priors and online experience through two coupled RL loops for learning high-level strategy and primitive control, linked by a shared goal–outcome interface. We instantiate this framework as PREINFORCE: a frozen foundation model constructs and revises an editable strategic program from online feedback, while a goal-conditioned neural policy learns fine-grained execution through temporal-difference contrastive learning. With 100,000 online interactions per game, PREINFORCE is, to our knowledge, the first agent to exceed the human reference in per-game mean score on all 26 Atari 100k games.

1 INTRODUCTION

Humans learn and refine their behavior through ongoing interaction with the world (Lake et al., 2017). How can artificial agents achieve a similar capacity for *online self-improvement*? Such improvement requires connecting what an agent already knows with what it learns through interaction, and translating both into effective action. This places three requirements on an agent: it must (i) exploit broad prior knowledge, (ii) adapt *sample-efficiently* from limited online experience, and (iii) execute fine-grained actions at the environment's native time scale.

Online reinforcement learning (RL) and foundation-model-based agents offer two routes toward this goal. RL learns primitive-action policies from environment feedback, but usually acquires task knowledge through interaction, making high-level strategy discovery difficult when rewards are sparse or delayed (Mnih et al., 2015; Schwarzer et al., 2023). Foundation-model-based agents draw on pretrained knowledge and can revise plans, memories, or programs from online feedback (Shinn et al., 2023; Wang et al., 2024a; Zhang et al., 2026), but often leave primitive control to predefined tools, skills, or control APIs (Figure 1, left and middle).

Atari games have played a central role in the development of deep RL (Bellemare et al., 2013; Machado et al., 2018). These games provide a concrete testbed for our question: strategic priors can guide long-horizon decisions, but acting on that knowledge requires fine-grained joystick control from visual observations. The modern Atari 100k protocol further tests whether agents can learn and improve sample-efficiently (Kaiser et al., 2020). Without task priors, exploration is often a central bottleneck for RL in these games. For example, an unguided agent in Private Eye must discover the prerequisite chain linking item collection and delivery. In Section 3, we quantify this bottleneck and show that unguided discovery can remain overwhelmingly unlikely even when previously discovered behaviors can be perfectly retained. On the other hand, foundation-model priors can supply strategic knowledge, but often leave primitive actions unspecified (Waytowich et al., 2024). For example, knowing which items to collect and where to deliver them in Private Eye does not specify the movements and timing required at each step. The central problem is how to turn strategic priors into action-level guidance that can efficiently improve through interaction.

In this work, we introduce *dual-loop agents*, a framework that unifies prior knowledge and online experience through coupled learning at two abstraction levels. Strategic priors may be incomplete, and execution capabilities change with learning. An upper RL loop combines priors with execution

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.

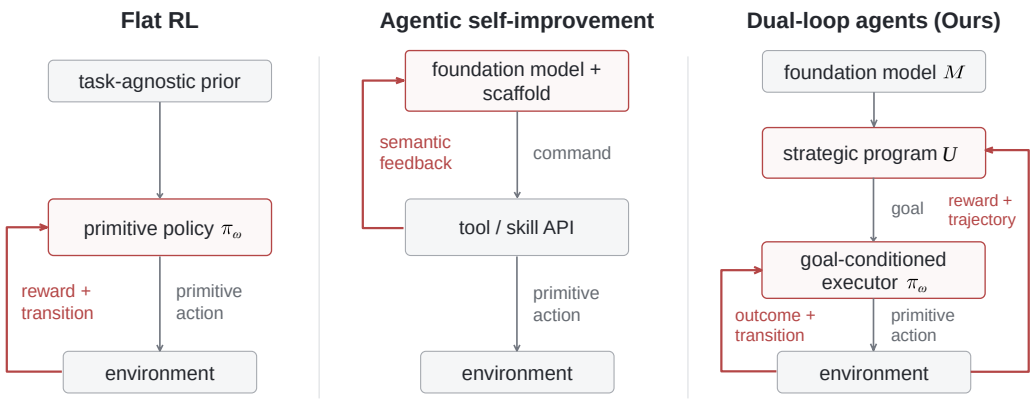

Figure 1: Online self-improvement in three agent designs. **Flat RL** and **agentic self-improvement** each typically use one learning loop to update primitive control and a high-level scaffold with a predefined execution interface, respectively. **Dual-loop agents** couple two RL loops for strategic and primitive-control learning through a goal–outcome interface. Trainable modules are in red.

feedback to revise a strategic policy; a lower RL loop learns goal-conditioned primitive control from the same experience. A shared goal–outcome interface connects the levels at an intermediate granularity: the upper policy specifies goals and labels observed outcomes for lower-level learning, and execution feedback informs subsequent strategic revision (Figure 1, right). This framework takes inspiration from multi-level learning in early hierarchical RL (Dayan & Hinton, 1992; Parr & Russell, 1997; Sutton et al., 1999), and we further make prior knowledge an explicit input to upper-level policy learning, combining foundation-model guidance with online experience.

Realizing this framework requires an upper policy that can incorporate priors and be revised efficiently online. Fine-tuning foundation models from limited online data can be inefficient and cause forgetting (Luo et al., 2023); in-context learning needs to handle a growing context, while per-step model calls incur inference latency (Yu et al., 2023; Black et al., 2025). We therefore keep the model frozen and use editable code as a persistent strategic policy (Liang et al., 2023), revised using priors and interaction feedback. Explicit primitive-control code can be brittle when it must handle precise timing and recovery (Weng, 2026). Our solution is to let the program select intermediate-granularity goals for a neural executor instead of prescribing fine-grained actions directly. At inference time, the program and executor produce actions *without* foundation-model calls.

The lower level faces another supervision problem for goal-reaching. Sparse task rewards may not indicate whether an intermediate goal was achieved, while synthesizing dense rewards using foundation models or code requires potentially complex specifications (Ma et al., 2024; Xie et al., 2024) and risks reward hacking when proxies are misspecified (Amodei et al., 2016). We therefore extend self-supervised goal-conditioned RL (Eysenbach et al., 2022; Zheng et al., 2024) from predicting future *state* occupancy to predicting program-defined *outcomes*. The program labels what occurred, and contrastive temporal-difference (TD) learning uses these labels to learn which actions make the desired outcome likely, without a hand-designed low-level reward or a separately trained task-specific reward model. We call the resulting prior-guided hierarchical RL algorithm PREINFORCE.

We evaluate PREINFORCE on Atari 100k, using GPT models prompted with game knowledge to construct and revise high-level programs. With 100,000 online interactions per game, PREINFORCE is, to our knowledge, the first agent to exceed the human reference in per-game mean score on all 26 games. In summary, our main contributions are:

- A dual-loop agent self-improvement framework that couples prior-guided strategy improvement and primitive-control learning through a shared goal–outcome interface.
- PREINFORCE, which instantiates this framework with editable program search and an extension of self-supervised goal-conditioned RL to program-defined outcomes, enabling fine-grained policy learning without a separately specified low-level reward.
- The first agent, to our knowledge, to achieve per-game mean scores above the human reference on all 26 Atari 100k games under a budget of 100,000 interactions per game.

2 Related Work

Sample-efficient RL on Atari. Atari games are among the longest-established benchmarks for RL agents (Bellemare et al., 2013; Machado et al., 2018). Early milestones include DQN (Mnih et al., 2015) and Rainbow (Hessel et al., 2018). More recent work targets limited-data learning under the Atari 100k protocol through world modeling and planning (Kaiser et al., 2020; Wang et al., 2024b; Alonso et al., 2024), representation learning (Laskin et al., 2020; Schwarzer et al., 2021), and regularized value learning (Schwarzer et al., 2023). Despite these advances, no existing RL method reaches the human reference on all Atari 100k games, motivating the use of task priors.

Hierarchical and goal-conditioned RL. Early hierarchical RL work introduced the notion of temporal and task abstraction (Dayan & Hinton, 1992; Parr & Russell, 1997; Sutton et al., 1999). Yet, a long-standing difficulty in hierarchical RL is to learn representations or interfaces bridging planning and control (Nachum et al., 2019; Jiang et al., 2019; Zhang et al., 2020; 2023a). We address this problem using an editable program to explicitly construct intermediate-granularity goals that bridge semantic priors and actions. Goal-conditioned values, hindsight relabeling, and contrastive learning support goal-reaching from experience (Schaul et al., 2015; Andrychowicz et al., 2017; Eysenbach et al., 2022; Zheng et al., 2024). We extend this formulation to program-defined outcomes, supporting self-supervised control in a compact and semantically meaningful representation space.

Prior-guided RL. Prior guidance has been used in RL for representation learning and action guidance (Chen et al., 2025; Yan et al., 2025), reward design (Ma et al., 2024; Xie et al., 2024), and exploration (Du et al., 2023). Language plans can also invoke pretrained skills, motion planners, or human-specified skills from AI-feedback rewards (Ichter et al., 2023; Dalal et al., 2024; Klissarov et al., 2025). On Atari, prior work adds object–agent interaction judgments as auxiliary rewards (Wu et al., 2023), or evaluates language agents with RAM-derived textual observations (Li et al., 2025). By contrast, PREINFORCE uses one general online learning procedure across all games under the standard Atari 100k protocol, without requiring additional human annotations or pretrained skills.

Self-improving agents. Recent work frames a deployed agent as a foundation model coupled to an operational scaffold that can be updated from experience (Ren et al., 2026; Zhang et al., 2026). Different approaches target inference-time refinement through self-feedback (Madaan et al., 2023), verbal reflection and memory (Shinn et al., 2023), accumulated experiential knowledge (Zhao et al., 2024), or agent co-evolution (Xia et al., 2026). This literature primarily targets high-level reasoning, tools, or skills, leaving the mapping to primitive actions to a predefined controller or API. PREINFORCE complements this line by combining an editable strategic program with an online-learning neural executor required for fine-grained and real-time interaction.

Programmatic policies. Programmatic policy search predates foundation models (Verma et al., 2018), while code-generating models support executable policies, skill libraries, and revision from feedback (Liang et al., 2023; Wang et al., 2024a). Code can encode strategic priors and support online policy improvement, but purely rule-based policies face practical expressivity limits and struggle to capture precise actions and context-dependent control (Weng, 2026). PREINFORCE instead uses code for goal selection and outcome labeling. This division lets prior knowledge guide control without requiring the complete control policy to be expressed and revised as explicit program logic.

3 Prior Guidance under a Finite Interaction Budget

In this section, we analyze the difficulty of discovering high-level strategies through RL without task priors in an abstract Atari model. The main abstraction is to represent a strategy as a sequence of task-relevant *events*. Each event models a milestone that may require many primitive actions, such as collecting an item or delivering it to a destination. We call each event's first successful completion its discovery. To isolate this exploration problem, here we consider an *idealized RL learner* with perfect retention of discovered events (i.e., no credit assignment problem). Therefore, the remaining discovery difficulty motivates action priors even when known behaviors can be executed effortlessly.

Formally, consider an episodic decision process with observations, actions, and rewards for completing events. The task requires completing m prerequisite events in an admissible first-discovery order from $\mathcal{O}$, with episode length H and total interaction budget B. An unknown event provides no reward for intermediate progress. Its local progress starts at zero and resets at episode boundaries.

After a discovery, the next unknown event can be attempted in the remaining episode time. When several next events are admissible, the learner selects one uniformly and retries it until discovery.

The idealized RL learner instantly acquires a perfect execution skill after an event's first discovery, retains it across episodes, and replays known prerequisites at no interaction cost. Its interaction budget is therefore spent only on first discovery. To model exploration of an unknown event, we abstract the agent's raw action space into three primitive actions a_-, a_0, a_+ with progress increments $-1, 0, +1$. Their probabilities form an *action prior* $\rho = (\rho_-, \rho_0, \rho_+)$, where $\rho_j \geq 0$ and $\sum_j \rho_j = 1$. We define the expected progress per action as $\Delta_{\text{prior}} \coloneqq \rho_+ - \rho_-$. Before discovery, the action distribution conditional on the interaction history is fixed at ρ. Extensions to history-dependent action distributions and arbitrary bounded increments are in Appendix A.1.

Let X_t be the progress increment at step t and $S_t = \sum_{k=0}^{t-1} X_k$, with $S_0 = 0$. An event requires reaching a net progress $S_t \geq L$, where L is a positive integer. The probability of discovering an event within H steps then admits the following drift bound.

Theorem 1 (Event discovery under an action prior)**.** *In the model above with* $[z]_+ = \max\{z, 0\}$,

$$\Pr_{\rho}(\textit{event discovery by } H) \leq q_\rho(L, H) \coloneqq \exp\left(-\frac{\left[L - H[\Delta_{\text{prior}}]_+\right]_+^2}{2H}\right). \tag{1}$$

The proof is in Appendix A.2. With a uniform action prior ($\Delta_{\text{prior}} = 0$), the bound decays as $\exp(-L^2/(2H))$, indicating that distant events are unlikely to be discovered through random fluctuations alone. Positive drift $\Delta_{\text{prior}} > 0$ can reduce the progress those fluctuations must supply.

For the idealized learner, let $N = \lceil B/H \rceil$. Each unknown event has at most N attempts of at most H steps, so its conditional discovery probability within the budget is at most $\min\{1, Nq_\rho(L, H)\}$. Composing these bounds over the m discoveries and admissible orders gives

$$\Pr_{\rho}(\text{complete the task by } B) \leq Q_\rho(B) \coloneqq \min\{1, |\mathcal{O}|\,[\min\{1, Nq_\rho(L, H)\}]^m\}. \tag{2}$$

Appendix A.3 gives the proof.

Case study: Private Eye. Private Eye is a driving-and-investigation game in which the player collects stolen objects, delivers them to designated locations, and catches the thief. This process could be abstracted as two pickup–delivery pairs followed by an arrest, with the two pairs acquired in either order: $m = 5$ and $|\mathcal{O}| = 2$. For each unknown event, we assume uniform exploration after every learning history, giving $\rho = u = (1/3, 1/3, 1/3)$ and $\Delta_{\text{prior}} = 0$. This models the absence of external task priors that provide directional guidance during event discovery, as in conventional RL.

Each event requires roughly a 30-second directed traversal at 60 emulator frames per second and an action repeat of four. This sets a common reference distance $L = 30 \times 60/4 = 450$. A standard three-minute episode gives $H = 180 \times 60/4 = 2700$. With $B = 100{,}000$, we have $N = \lceil B/H \rceil = 38$. Evaluating Equation 2 under uniform exploration gives

$$\Pr_{u}(\text{complete the task by } B) \leq Q_u(B) \approx 5.88 \times 10^{-74}. \tag{3}$$

That is, discovery remains overwhelmingly unlikely even when previously discovered behaviors are retained and replayed perfectly. Also, a moderate action prior can substantially improve discovery. For example, under the same budget, shifting 18 percentage points of action probability from zero to forward progress gives a success probability of at least 0.920 (Appendix A.4).

From action priors to goal-guided control. Although the above analysis demonstrates the value of action-level priors, prior knowledge in practice often specifies high-level strategies, such as destinations and prerequisites, without supplying primitive-action probabilities. The next section connects these two levels through goals: a strategic policy uses priors to select intermediate outcomes, and a controller learns the actions that realize them.

4 Dual-Loop Agents and the PREinforce Algorithm

Section 3 motivates action-level guidance, but leaves open how strategic priors should produce it. In this section, we introduce the dual-loop agent framework, which adopts a goal–outcome interface to

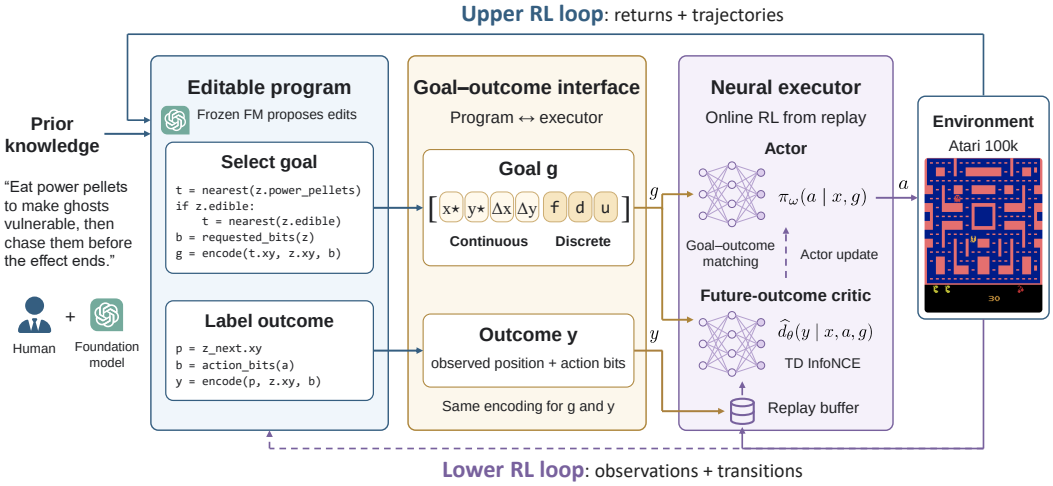

Figure 2: The PREINFORCE algorithm. The upper RL loop revises a prior-guided program from returns and trajectories. Its goals and outcome labels supervise the lower self-supervised contrastive RL loop, whose interaction data also inform program revision.

mediate this influence. Our main idea is that for state x and a goal-conditioning variable g satisfying $p(a \mid x, g) = p(a \mid x, g, \text{prior})$, the prior-guided policy can be decomposed as a hierarchical policy

$$p(a \mid x, \text{prior}) = \int \underbrace{p(a \mid x, g)}_{\text{lower-level policy}} \underbrace{p(g \mid x, \text{prior})}_{\text{upper-level policy}} \, \mathrm{d}g. \tag{4}$$

An important point here is that we can pick a goal variable g with *intermediate granularity* by representing a short-horizon state transition, e.g., reaching a nearby state. Such goal representations preserve semantics that could be directly connected to high-level priors, while also reducing the exploration horizon for lower-level RL. Goals also provide a route for prior knowledge to guide control: given x and g, the executor receives no further prior input. In Appendix A.5, we show provable benefits of using goals as a form of *event-splitting* in our abstract Atari model.

Based on the decomposition in equation 4, the dual-loop agents framework learns both policies from interaction using two RL loops. The upper RL loop combines priors with experience to revise goal selection, maximize return, and adapt to evolving low-level execution capabilities. The lower RL loop learns to realize the selected goals through primitive actions.

The PREINFORCE algorithm (Figure 2) instantiates this framework with a frozen foundation model M, an editable program U, and a goal-conditioned neural policy π_ω. The program contains visual state estimation, goal selection, and outcome labeling. The foundation model proposes program updates from priors and online experience. Both loops use the same interaction stream: returns and trajectories inform program revision, and program-defined outcome labels turn those trajectories into supervision for the executor. Revisions can change which goals are pursued and how outcomes are recognized, while execution feedback informs subsequent strategic choices. At inference, action execution uses only the program and the learned policy *without* the foundation model. We next define the interface, the two RL loops, and their synchronization.

4.1 PROBLEM FORMULATION AND THE GOAL–OUTCOME INTERFACE

We consider visual RL with observations o_t, primitive actions $a_t \in \mathcal{A}$, and environment rewards r_t. The program U maintains a structured state z_t from the observation, action, and reward history. Its goal selector h chooses $g_t = h(z_{\leq t})$, and its outcome labeler κ assigns $y_{t+1} = \kappa(z_t, a_t, z_{t+1})$ to each transition. The executor samples $a_t \sim \pi_\omega(\cdot \mid x_t, g_t)$ during training and takes the greedy action during evaluation. Here x_t is the executor's state representation, distinct from the program state z_t. We assume x_t is sufficient for the dynamics and outcome labels in the formulation below; the implementation approximates it with stacked images.

Goals specify intended outcomes, while labels describe observed outcomes. Both use a shared encoding with continuous position coordinates and discrete instruction components in our Atari

implementation (Appendix C.1). Positions are inferred from raw visual observations, and achieved instruction bits are read from executed actions, without privileged information such as RAM.

4.2 Upper RL Loop: Prior-Guided Program Search

Given an executor π_ω, the upper loop searches over programs U to maximize the environment return

$$J_{\text{upper}}(U) = \mathbb{E}_{\tau \sim p_{U,\pi_\omega}} \left[\sum_{t=0}^{T-1} \sum_{a \in \mathcal{A}} \pi_\omega(a \mid x_t, h(z_{\le t}))\, R(x_t, a) \right], \tag{5}$$

where T is the trajectory length, $R(x, a)$ is the expected immediate environment reward, and p_{U,π_ω} is the trajectory distribution induced by the program and executor.

A frozen foundation model M combines pretrained knowledge and task-specific priors supplied in human-written prompts with online experience to construct and revise U, which comprises four functional components. **Visual perception** integrates observations to identify and track the key objects, recover scene geometry, and detect task-relevant events. **Motion estimation and prediction** estimates velocities and motion patterns from observed transitions, predicting player movement and the trajectories of relevant moving objects. **Task and goal planning** combines task priors with the perceived state and motion predictions to select tasks, routes, and local goals, using task memory and cross-scene connectivity as needed. **Outcome labeling** maps causal perceptual readouts and executed actions into achieved outcomes under the shared goal–outcome semantics.

The PREInforce algorithm specifies these components and interfaces and uses M to generate and revise their game-specific implementations (see Appendix B for details). Note that while the program must ground the goal, it need not specify a goal-reaching reward function, alleviating the difficulty of *reward specification* for complex behaviors (Hadfield-Menell et al., 2017).

With foundation-model guidance, the upper RL loop can use more diverse forms of interaction data than conventional RL to guide program revisions. In practice, we use the current code, task priors, and accumulated development evidence $\mathcal{E}$, including returns, gameplay frames, executed actions, and diagnostics output by the foundation model itself. We organize revision into two foundation-model roles: a *Developer*, which constructs and revises the program, and an *Auditor*, which independently reviews gameplay before inspecting the code. Audit reports enter $\mathcal{E}$ to guide subsequent revisions. Appendix B provides details and prompt templates.

4.3 Lower RL Loop: Future-Outcome Prediction

For the lower RL loop, we extend the future-state occupancy formulation of contrastive RL (Eysenbach et al., 2022; Zheng et al., 2024) to program-defined outcomes. The key idea is that each labeled outcome provides a target for learning the *achieved behavior* in the program-defined semantic space, even when the requested goal is missed or no task reward is received. This enables self-supervised RL for primitive control using TD learning with no explicit reward function.

Future-outcome distribution. For a fixed program, policy, and goal, let $\mathcal{Y}$ be the outcome space and $p_k^\pi(y \mid x, a, g)$ the distribution of the outcome at the kth subsequent transition. We define the *discounted outcome occupancy* as

$$d^\pi(y \mid x, a, g) \coloneqq (1-\gamma) \sum_{k=1}^{\infty} \gamma^{k-1} p_k^\pi(y \mid x, a, g), \tag{6}$$

where $\gamma \in (0, 1)$ is a fixed discount. Taking outcomes to be *next states* recovers the usual state-occupancy formulation (Dayan, 1993; Zheng et al., 2024). Using the *next outcome* instead provides a more compact control-relevant representation: visually different transitions may share the same outcome, which need not be inferred from the next state alone. Under the state-sufficiency assumption above, the program labeler κ induces an immediate-outcome distribution $K(\cdot \mid x, a, x')$ conditional on the executor transition. For a transition kernel P with $x' \sim P(\cdot \mid x, a)$ and $a' \sim \pi(\cdot \mid x', g)$, the outcome Bellman equation is given by

$$d^\pi(y \mid x, a, g) = (1-\gamma)\mathbb{E}_{x'}[K(y \mid x, a, x')] + \gamma \mathbb{E}_{x', a'}[d^\pi(y \mid x', a', g)]. \tag{7}$$

Critic learning with TD InfoNCE. Based on the decomposition of Equation 7, the critic $f_\theta(x, a, g, y)$ scores candidate outcomes to estimate the outcome occupancy using TD learning with

a contrastive InfoNCE loss. For a reference distribution μ over outcomes in replay, define

$$\widehat{d}_\theta(y \mid x, a, g) \coloneqq \frac{\mu(y) \exp f_\theta(x, a, g, y)}{\mathbb{E}_{\tilde{y}\sim\mu}[\exp f_\theta(x, a, g, \tilde{y})]}. \tag{8}$$

This normalized model represents the outcome distribution on the support of μ. For a replay transition and its outcome label $y^+ \sim K(\cdot \mid x, a, x')$, we minimize the population critic loss

$$\mathcal{L}_{\text{critic}}(\theta) = -\mathbb{E}\big[(1-\gamma)\log \widehat{d}_\theta(y^+ \mid x, a, g) + \gamma w_{\bar{\theta}}(x', a', g, y) \log \widehat{d}_\theta(y \mid x, a, g)\big]. \tag{9}$$

Here the expectation is over replay tuples from $\rho_{\mathcal{D}}$, $a' \sim \pi_\omega(\cdot \mid x', g)$, and $y \sim \mu$. The replay tuples include transitions, labels, and relabeled goals, and $w_{\bar{\theta}} = \widehat{d}_{\bar{\theta}}/\mu$ is a target-critic density ratio held fixed during each critic update. Under certain assumptions, it can be shown that the above population update has Equation 6 as its fixed point. In practice, our implementation also uses an auxiliary contrastive term to handle the discrete instruction components. See Appendices C.2–C.3 for the fixed-point argument and finite-batch implementation.

Actor learning by goal–outcome matching. The actor trains on goal–outcome pairs relabeled from observed future states and executed actions. For a minibatch $\mathcal{B} = \{(x_i, g_i, y_i^\star)\}_{i=1}^N$, $y_i^\star = y^\star(g_i)$ denotes the target outcome corresponding to g_i (Appendix C.1). The actor maximizes

$$J_{\text{actor}}(\omega) = \mathbb{E}_{\mathcal{B}}\left[\frac{1}{N}\sum\nolimits_i \left(\sum\nolimits_{a\in\mathcal{A}} \pi_\omega(a \mid x_i, g_i) s_{\theta,ii}(a) + \alpha \mathcal{H}(\pi_\omega(\cdot \mid x_i, g_i))\right)\right], \tag{10}$$

where $s_{\theta,ij}(a) = f_\theta(x_i, a, g_i, y_j^\star) - \log \sum_{m=1}^N \exp f_\theta(x_i, a, g_i, y_m^\star)$ is the normalized critic score, and $\alpha \geq 0$ controls the policy-entropy bonus. See Appendix C.3 for details.

4.4 Synchronizing Upper and Lower RL

We first construct and validate U using priors and initial environment interactions, then train the executor with U fixed. The two RL loops subsequently learn from shared interaction data but update at different frequencies. Specifically, joint training proceeds in rounds. At the start of each round, we freeze a copy of the current actor for execution. Each round contains trials, each collecting interactions under a fixed program–executor pair. Between trials, M revises the program using accumulated feedback. During trials, a separate learner repeatedly updates the actor and critic from replay, while the execution copy remains fixed. The updated actor is used for execution in the next round. At the end of development, the program–executor pair last confirmed by the Auditor is frozen for evaluation. Algorithm 1 summarizes the procedure.

5 Experiments

In this section, we evaluate PREinforce on Atari 100k, provide ablations to assess the contributions of our upper and lower RL loops, and present case studies to illustrate the effect of incorporating strategic priors in solving tasks that are difficult for conventional RL.

5.1 Setup and Main Results

Benchmark and baselines. We evaluate on the 26-game Atari 100k benchmark, with a budget of 100,000 environment interactions per game. The agent receives image observations and environment rewards under the standard Atari protocol, along with human-supplied game knowledge provided to the foundation model through natural-language prompts. For baselines, we select BBF (Schwarzer et al., 2023) and EfficientZero V2 (Wang et al., 2024b) as representative state-of-the-art model-free and model-based RL methods, respectively, using their published per-game scores. We also include an RL reference that takes the highest published mean for each game, and program-only code policies that directly output primitive actions via code (Weng, 2026), evaluated using the same game-knowledge prompts as PREinforce. Details and broader comparisons appear in Appendix D.

Evaluation and metrics. Let S_i be the mean evaluation score for game i. We use the human-normalized score $\text{HNS}_i \coloneqq (S_i - S_i^{\text{random}})/(S_i^{\text{human}} - S_i^{\text{random}})$, where S_i^{random} and S_i^{human} are the reference scores of the random policy and humans, respectively. We report the mean and

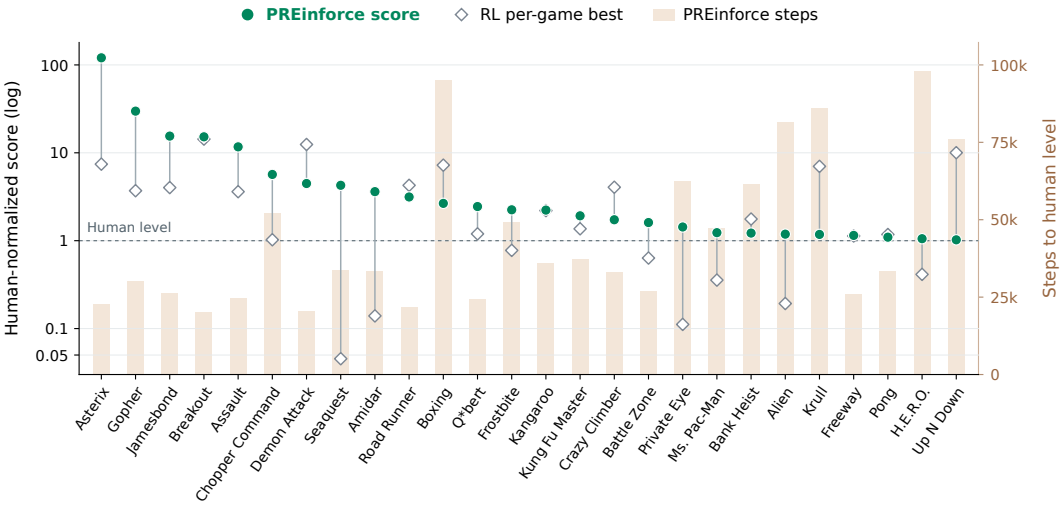

Figure 3: Atari 100k results ordered by PREINFORCE mean HNS. Points show per-game mean HNS; bars show PREINFORCE's interaction steps to human level. The RL reference is the per-game highest published mean among the Atari 100k RL agents (Appendix D). The dashed line marks human performance (HNS $= 1$). The HNS axis uses a logarithmic scale.

Method	Mean HNS	IQM	Above human
BBF	2.247	1.139	12/26
EfficientZero V2	2.680	1.305	15/26
Code policies	3.649	1.443	14/26
PREINFORCE	**9.202**	**2.445**	**26/26**
w/o upper RL loop	0.534	0.024	2/26

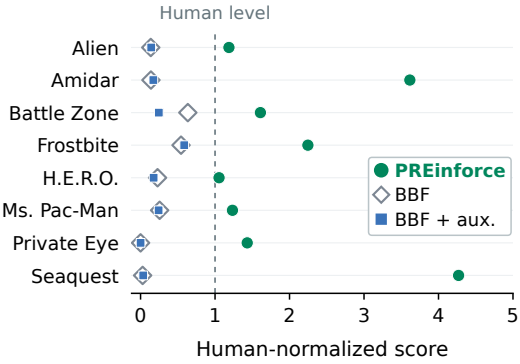

Figure 4: **Left:** aggregate Atari 100k results of PREINFORCE and baselines. **Right:** auxiliary-reward comparison on eight games. The dashed line marks human-level performance (HNS $= 1$).

interquartile mean (IQM) of the 26 per-game mean HNS values, and *Above human*: the number of games with $\text{HNS}_i > 1$, reported out of 26. For PREINFORCE, we also report the per-game interaction steps needed to reach human level, determined by evaluating saved program–executor pairs and identifying the earliest pair whose mean exceeds the human reference.

Main results. As shown in Figures 3 and 4, PREINFORCE exceeds the human reference in per-game mean score on all 26 games, compared with 12/26 for BBF, 15/26 for EfficientZero V2, and 14/26 for Code policies. Its mean HNS and IQM also surpass all baselines. In 21/26 games, at most 75k steps suffice for PREINFORCE to reach human level. Compared with the RL reference, PREINFORCE achieves higher scores in 18/26 games, with a particular advantage in games requiring long-term strategies, such as Private Eye, Amidar, Seaquest, and Ms. Pac-Man.

5.2 Ablation Studies

We assess the upper and lower RL loops of PREINFORCE through two ablations (Figure 4). **First,** we freeze the initial validated program while retaining executor learning under the same 100k-step budget. This variant substantially underperforms PREINFORCE and reaches the human reference in only 2/26 games. This shows that *the initial prior-guided program alone is insufficient across games*, supporting the role of online self-improvement. **Second,** as an alternative prior-integration comparison, we compare PREINFORCE's hierarchical goal-conditioned control with a flat BBF agent trained using environment rewards plus auxiliary rewards produced by the foundation model (Appendix D.5). In particular, we allow the foundation model to write reward functions from the upper-

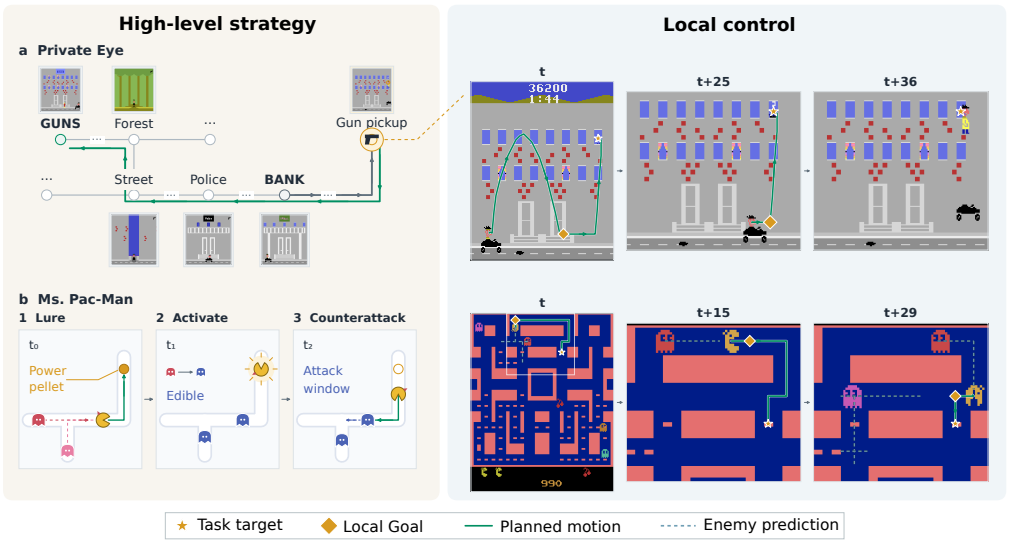

Figure 5: Prior-guided high-level strategies and local control in PREINFORCE on two representative games. **Top:** long-term route planning with local goals guiding jumps in Private Eye. **Bottom:** power-pellet and counterattack strategy with local goals guiding movement in Ms. Pac-Man.

level program of PREINFORCE, selected *after* 100k online interactions. We evaluate this method on the eight games where the per-game RL reference remains below human level. The results show that it yields negligible gains compared to standard BBF, suggesting that *translating priors directly into rewards remains challenging*, while explicit goal conditioning provides more guidance for control.

5.3 CASE STUDIES

We select Private Eye and Ms. Pac-Man, two representative games where conventional RL performs substantially below the human level, to illustrate how PREINFORCE combines prior-guided high-level planning with local control. Figure 5 visualizes the learned strategies and control behaviors.

Private Eye: Prerequisites and route planning. The program organizes gameplay into a sequence of subtasks across scenes, using prerequisites and task progress to select the next objective. Scene connections support route planning between locations, such as collecting a gun and returning it to the gun shop, while local goals guide its execution. The lower-level policy learns to follow these goals by jumping over ground hazards and reaching elevated items such as the gun.

Ms. Pac-Man: Power-pellet strategy. The program coordinates pellet collection and counterattacks on ghosts through a timed sequence: lure ghosts closer, consume a power pellet, and counterattack while the ghosts are edible. The planner combines these task choices with movement estimates and predicted ghost trajectories to select routes and local goals. The lower-level policy learns to follow these goals and advance along a route toward the power pellet as nearby threats change.

6 DISCUSSION AND LIMITATIONS

Dual-loop agents offer a general way to combine prior knowledge with experience through learning at multiple abstraction levels. We believe that the framework could extend beyond Atari to open-ended games and embodied tasks, where agents must revise high-level strategies while also acquiring low-level execution skills. A natural extension is to let upper-level code selectively invoke pretrained perceptual models for visual inputs that are difficult to handle with explicit code rules.

Limitations. To extend PREINFORCE to a broader range of settings, an important future direction is to construct more general goal–outcome interfaces that are both expressive and flexible. Open problems include choosing useful goal representations to handle skills that span multiple granularities and may evolve over time, and grounding outcomes under partial observability. The two learning loops must also remain coordinated as strategies, perception, and control capabilities change, which may call for novel training algorithms that build on the dual-loop agent framework.

AI USE STATEMENT

The Atari implementation uses GPT models, prompted with human-supplied game knowledge, to construct and revise strategic programs. Generative AI tools also assisted in refining the idealized event-discovery model and deriving the stopping-time and martingale arguments used in the proofs. The authors also used AI tools to assist with literature searches, manuscript drafting and editing, and figure preparation. The core research methodology and experimental design were developed by the authors. Generative AI tools assisted with implementing the methods and experimental code, which the authors reviewed and validated. The authors take responsibility for the final content, including verification of the mathematical arguments, calculations, references, code, and empirical claims.

ETHICS STATEMENT

This work evaluates reinforcement learning agents in simulated Atari environments and involves neither human-subject experiments nor personal data.

REPRODUCIBILITY STATEMENT

The appendices detail the learning procedure, code interfaces, executor settings, hyperparameters, computational costs, and token usage. The supplementary material includes the implementation, prompts, and human-supplied knowledge for each game.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in Atari. In *Advances in Neural Information Processing Systems*, volume 37, pp. 58757–58791, 2024.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025.

William Chen, Oier Mees, Aviral Kumar, and Sergey Levine. Vision-language models provide promptable representations for reinforcement learning. *Transactions on Machine Learning Research*, 2025.

Murtaza Dalal, Tarun Chiruvolu, Devendra Singh Chaplot, and Ruslan Salakhutdinov. Plan-Seq-Learn: Language model guided RL for solving long horizon robotics tasks. In *International Conference on Learning Representations*, 2024.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5, 1992.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G. Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *International Conference on Learning Representations*, 2023.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8657–8677. PMLR, 2023.

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan R. Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 35603–35620, 2022.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023.

Yiding Jiang, Shixiang Gu, Kevin P. Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020.

Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C. Machado, and Pierluca D'Oro. MaestroMotif: Skill design from artificial intelligence feedback. In *International Conference on Learning Representations*, 2025.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5639–5650. PMLR, 2020.

Wenhao Li, Wenwu Li, Chuyun Shen, Junjie Sheng, Zixiao Huang, Di Wu, Yun Hua, Wei Yin, Xiangfeng Wang, Hongyuan Zha, and Bo Jin. TextAtari: 100K frames game playing with language agents. *arXiv preprint arXiv:2506.04098*, 2025.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation*, pp. 9493–9500, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2019.

Ronald Parr and Stuart J. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10, pp. 1043–1049, 1997.

Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. Self-improvements in modern agentic systems: A survey. *arXiv preprint arXiv:2607.13104*, 2026.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320. PMLR, 2015.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021.

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 *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 30365–30380. PMLR, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5045–5054. PMLR, 2018.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Shengjie Wang, Shaohuai Liu, Weirui Ye, Jiacheng You, and Yang Gao. EfficientZero V2: Mastering discrete and continuous control with limited data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51041–51062. PMLR, 2024b.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1995–2003. PMLR, 2016.

Nicholas R. Waytowich, Devin White, MD Sunbeam, and Vinicius G. Goecks. Atari-GPT: Benchmarking multimodal large language models as low-level policies in Atari games. *arXiv preprint arXiv:2408.15950*, 2024.

Jiayi Weng. Learning beyond gradients, May 2026. URL `https://trinkle23897.github.io/learning-beyond-gradients/`. Blog post. Accessed 2026-09-08.

Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M. Mitchell. Read and reap the rewards: Learning to play Atari with the help of instruction manuals. In *Advances in Neural Information Processing Systems*, volume 36, pp. 1009–1023, 2023.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. In *Conference on Language Modeling*, 2026.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations*, 2024.

Xue Yan, Yan Song, Xidong Feng, Mengyue Yang, Haifeng Zhang, Haitham Bou-Ammar, and Jun Wang. Efficient reinforcement learning with large language model priors. In *International Conference on Learning Representations*, 2025.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering Atari games with limited data. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25476–25488, 2021.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 374–404. PMLR, 2023.

Genrui Zhang, Erle Zhu, Jinfeng Zhou, Caiyan Jia, and Hongning Wang. SkillEvolver: Skill learning as a meta-skill. *arXiv preprint arXiv:2605.10500*, 2026.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21579–21590, 2020.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Adjacency constraint for efficient hierarchical reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4152–4166, 2023a.

Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. STORM: Efficient stochastic transformer based world models for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Chongyi Zheng, Ruslan Salakhutdinov, and Benjamin Eysenbach. Contrastive difference predictive coding. In *International Conference on Learning Representations*, 2024.

A PRIOR-DEPENDENT DISCOVERY: PROOFS AND NUMERICAL ANALYSIS

We prove the discovery bounds in Section 3 and evaluate action priors and intermediate goals under its retained-event model. Throughout the numerical analysis, discovered behaviors are retained and replayed for free, unfinished progress resets only at episode boundaries, and all discoveries share the same interaction budget.

A.1 GENERAL BOUND FOR ADAPTIVE ACTION DISTRIBUTIONS

We first establish the general bound from which the main-text result follows. Let $\mathcal{F}_t$ contain the interaction history and learner state before action a_t, with conditional action distribution $p_t(a \mid \mathcal{F}_t)$. For an adapted scalar progress process S_t with $S_0 = 0$, define $\tau_L = \inf\{t \geq 0 : S_t \geq L\}$, where event completion requires reaching $L > 0$. An action's expected increment and the resulting policy drift are

$$\begin{aligned} d_t(a) &\coloneqq \mathbb{E}[S_{t+1} - S_t \mid \mathcal{F}_t, a_t = a], \\ \mu_t(p) &\coloneqq \sum_{a \in \mathcal{A}} p_t(a \mid \mathcal{F}_t) d_t(a). \end{aligned} \tag{11}$$

These quantities may depend on the full history. Unlike the unit-step model, the increments may take arbitrary bounded real values.

Theorem 2 (General event-discovery bound)**.** *Let $H \geq 1$ be an interaction horizon. Suppose that, before τ_L, $|S_{t+1} - S_t| \leq c$ almost surely for a constant $c > 0$, and that a deterministic $D_p \geq 0$ bounds the cumulative conditional drift:*

$$\max_{0 \leq t \leq H \wedge \tau_L} \sum_{k=0}^{t-1} \mu_k(p) \leq D_p \qquad \textit{almost surely}. \tag{12}$$

Then, writing $[u]_+ = \max\{u, 0\}$,

$$\Pr_p(\textit{event completed by } H) \leq \Pr_p(\tau_L \leq H) \leq \exp\left(-\frac{[L - D_p]_+^2}{2Hc^2}\right). \tag{13}$$

The action distribution may be nonuniform and may change with the interaction history and online learning.

Proof. The argument follows the discovery event from first arrival to a probability bound. We work under the law induced by p_t, including any changes caused by online learning.

Step 1: preserve every first arrival. The original process can reach L and later fall below it, so its terminal position alone does not record every discovery. Instead, freeze it at τ_L. Let $\widetilde{\xi}_t$ denote its increments, $\widetilde{\mu}_t$ their conditional means, and $\widetilde{A}_t$ the accumulated conditional drift:

$$\begin{aligned} \widetilde{S}_t &= S_{t \wedge \tau_L}, & \widetilde{\xi}_t &= \widetilde{S}_{t+1} - \widetilde{S}_t, \\ \widetilde{\mu}_t &= \mathbb{E}[\widetilde{\xi}_t \mid \mathcal{F}_t] = \mathbf{1}\{t < \tau_L\}\mu_t(p), & \widetilde{A}_t &= \sum_{k=0}^{t-1} \widetilde{\mu}_k. \end{aligned} \tag{14}$$

Since τ_L is a stopping time, $\mathbf{1}\{t < \tau_L\}$ is $\mathcal{F}_t$-measurable. The increments satisfy $|\widetilde{\xi}_t| \leq c$, including at the first crossing. On a successful path, the stopped position stays at or above L.

Step 2: separate directional drift from random fluctuations. Subtracting the accumulated conditional drift gives $M_t = \widetilde{S}_t - \widetilde{A}_t$. Its expected next increment is zero:

$$\mathbb{E}[M_{t+1} - M_t \mid \mathcal{F}_t] = \mathbb{E}[\widetilde{\xi}_t - \widetilde{\mu}_t \mid \mathcal{F}_t] = 0.$$

Thus M_t is a martingale, even when action choices depend on the entire history. The remaining question is how likely these fluctuations are to supply the progress not covered by drift.

Step 3: control the size of those fluctuations. For any $\lambda > 0$, the conditional form of Hoeffding's lemma gives

$$\mathbb{E}[\exp\{\lambda(M_{t+1} - M_t)\} \mid \mathcal{F}_t] \leq \exp(\lambda^2 c^2/2). \tag{15}$$

The constant uses the interval width $2c$ of $\widetilde{\xi}_t$; centering preserves this width. It follows that

$$Z_t = \exp\left(\lambda M_t - \frac{\lambda^2 c^2 t}{2}\right)$$

is a nonnegative supermartingale with $Z_0 = 1$.

Step 4: evaluate the process when discovery occurs or time runs out. Let $T = \tau_L \wedge H$. Since $T \le H$, bounded optional stopping gives $\mathbb{E}[Z_T] \le 1$. On $\{\tau_L \le H\}$, Equation 12 implies $M_T = S_{\tau_L} - \widetilde{A}_{\tau_L} \ge L - D_p$, and therefore

$$Z_T \ge \exp\left(\lambda(L - D_p) - \frac{\lambda^2 c^2 H}{2}\right).$$

Every successful path therefore contributes at least this amount to $\mathbb{E}[Z_T]$, while unsuccessful paths contribute nonnegative amounts. Consequently,

$$\begin{aligned} 1 &\ge \mathbb{E}[Z_T] \\ &\ge \Pr_p(\tau_L \le H) \exp\left(\lambda(L - D_p) - \frac{\lambda^2 c^2 H}{2}\right). \end{aligned} \tag{16}$$

When $D_p < L$, setting $\lambda = (L - D_p)/(Hc^2)$ yields

$$\Pr_p(\tau_L \le H) \le \exp\left(-\frac{(L - D_p)^2}{2Hc^2}\right).$$

When $D_p \ge L$, use the upper bound of one. Event completion requires crossing L, so the same bound applies to completion. The proof includes all first-arrival times through T; it never sums a separate bound over each possible arrival time. This is why there is no additional factor H. □

A.2 Proof of Theorem 1

Proof. Under the fixed action prior ρ, the conditional expected increment before discovery is $\Delta_{\text{prior}} = \rho_+ - \rho_-$ and each increment lies in $[-1, 1]$. The cumulative conditional drift at time $t \le H \wedge \tau_L$ is $t\Delta_{\text{prior}}$. Its maximum is therefore at most

$$D_\rho = H[\Delta_{\text{prior}}]_+.$$

Applying Theorem 2 with $c = 1$ and this D_ρ gives Equation 1. The positive part accounts for the zero cumulative drift at $t = 0$, including when $\Delta_{\text{prior}} < 0$. □

A.3 Task Completion with Retained Discoveries

We derive Equation 2 for the idealized learner in Section 3, allowing the history-dependent progress laws of Appendix A.1. Fix an admissible discovery order o. Let T_i be the time at which the first i prerequisites in that order have been discovered, with $T_0 = 0$ and $T_i = \infty$ if discovery never occurs. Perfect retention and free replay of known prerequisites allow the learner to focus on event i after T_{i-1}. An attempt on this unknown event begins either at this time or at an episode boundary, with its local progress reset to zero. It ends upon discovery or at the next episode boundary. Thus it has at most H interactions, and each event has at most $N = \lceil B/H \rceil$ attempts within the total budget.

Assume that, conditional on every history at the start of an attempt on event i, its as-yet-undiscovered progress process satisfies Theorem 2 with deterministic bounds L_i, c_i, D_i and horizon H. This condition includes knowledge acquired from other events and unsuccessful attempts. Write

$$q_i = \exp\left(-\frac{[L_i - D_i]_+^2}{2Hc_i^2}\right), \qquad b_i = \min\{1, Nq_i\}.$$

Conditional on $\mathcal{F}_{T_{i-1}}$ on $\{T_{i-1} \le B\}$, a union bound over at most N attempts gives

$$\Pr(T_i \le B \mid \mathcal{F}_{T_{i-1}}) \le b_i.$$

This generously allows every event the full horizon on every attempt and the full episode count even if preceding discoveries used part of the budget. Since $\{T_i \le B\} \subseteq \{T_{i-1} \le B\}$, the tower property yields

$$\begin{aligned} \Pr(T_i \le B) &\le b_i \Pr(T_{i-1} \le B), \\ \Pr(T_m \le B) &\le \prod_{i=1}^{m} b_i. \end{aligned} \tag{17}$$

No independence between event discoveries is required. Crucially, q_i is used only before event i is first discovered, never for its later execution. For multiple admissible first-discovery orders, assume the same conditional bounds hold for the corresponding next unknown event after each order prefix. Summing the order-specific bounds gives

$$\Pr(\text{complete the task by } B) \le \min\left\{1, \sum_{o\in\mathcal{O}} \prod_{i=1}^{m} \min\{1, N q_{o,i}\}\right\}. \tag{18}$$

The modeled discovery orders must cover all ways of acquiring the required events; a restriction on final execution order alone would not suffice. With common L and fixed prior ρ, we have $q_{o,i} = q_\rho(L, H)$ for every order and stage. Equation 18 then reduces to Equation 2 in the main text.

A.4 Action Priors and Numerical Illustration

For the general model, compare p_t with any reference action distribution q_t at the same history. Adding and subtracting its expected increment gives

$$\mu_t(p) = \mu_t(q) + \underbrace{\sum_{a\in\mathcal{A}} [p_t(a \mid \mathcal{F}_t) - q_t(a \mid \mathcal{F}_t)] d_t(a)}_{\delta_t^{\text{prior}}}. \tag{19}$$

Here δ_t^{prior} is the progress gain relative to q_t. Define $d_t(a)$ through the environment kernel, including for actions of zero probability under p_t. Both distributions are evaluated at the same history.

For $p_\alpha = (1-\alpha)q + \alpha r$ and history-independent $d(a)$, let q be the reference distribution, r the prior, and $\alpha \in [0, 1]$. Define $\mu(q) = \sum_a q(a)d(a)$ and $\delta_{\text{prior}} = \sum_a [r(a) - q(a)]d(a)$. Then

$$\mu(p_\alpha) = \sum_a [(1-\alpha)q(a) + \alpha r(a)]d(a) = \mu(q) + \alpha\delta_{\text{prior}}.$$

In the main-text unit-step model with $q = u$ and $r = \rho$, $\mu(q) = 0$ and $\delta_{\text{prior}} = \rho_+ - \rho_- = \Delta_{\text{prior}}$: the relative gain equals the main-text expected progress for this zero-drift reference. Before stopping, the cumulative drift at time t is $t\mu(p_\alpha)$. Its maximum over $0 \le t \le H \wedge \tau_L$ is at most $H[\mu(p_\alpha)]_+$. Substitution into Theorem 2 gives

$$\Pr_{p_\alpha}(\tau_L \le H) \le \exp\left(-\frac{\left[L - H[\mu(q) + \alpha\delta_{\text{prior}}]_+\right]_+^2}{2Hc^2}\right). \tag{20}$$

The positive part is required because cumulative drift is zero at $t = 0$, even if the single-step drift is negative.

For the three-increment model, $\mu = p_+ - p_-$ and $c = 1$, giving

$$\Pr_{p}(\tau_L \le H) \le \exp\left(-\frac{\left[L - H[p_+ - p_-]_+\right]_+^2}{2H}\right). \tag{21}$$

The equal-probability case has zero drift. The resulting bound $\exp(-L^2/(2H))$ controls the first crossing directly, without the factor H introduced by a union bound over separate time points.

Shared numerical setting. We use exactly the Private Eye abstraction in Section 3: $m = 5$ events, $L = 450$, $H = 2700$, $B = 100{,}000$, and two admissible discovery orders. When both orders are available, the learner chooses uniformly which pickup–delivery pair to acquire first, then follows the selected order. Each unknown event has the same conditional progress law, regardless of its identity or preceding learning history. The budget contains 37 full episodes and a final 100-step partial episode. The upper bound allows $N = 38$ full-length attempts per event; the lower bound below uses only the 37 full episodes. Both bounds apply to this same interaction protocol.

Progress is net displacement,

$$S_t = \sum_{j=0}^{t-1} X_j = \#\{\text{forward moves before } t\} - \#\{\text{backward moves before } t\}. \tag{22}$$

Consequently, reaching L does not require a prescribed sequence of L forward actions: paths may include pauses and backward moves. The uniform baseline assumes uniform increments before each discovery after every learning history.

Zero-to-forward action priors. For each unknown event, draw actions from

$$\rho_\epsilon = \left(\frac{1}{3}, \frac{1}{3} - \epsilon, \frac{1}{3} + \epsilon\right), \qquad 0 \le \epsilon \le \frac{1}{3}. \tag{23}$$

This transfers ϵ probability mass from zero to forward progress, leaves the probability of a backward action unchanged, and gives $\Delta_{\text{prior}} = \epsilon$.

Evaluating the upper bound. Substituting $\Delta_{\text{prior}} = \epsilon$ into Equations 1–2 gives

$$q_\epsilon = \exp\left(-\frac{[L - H\epsilon]_+^2}{2H}\right), \qquad Q_\epsilon = \min\left\{1, 2[\min\{1, Nq_\epsilon\}]^5\right\}. \tag{24}$$

For uniform exploration, $q_0 \approx 5.18 \times 10^{-17}$ and $Q_0 \approx 5.88 \times 10^{-74}$, exactly the bound evaluated in the main text. For $\epsilon \ge 0.12$, the task upper bound becomes one; this alone does not establish improved success.

A complementary success lower bound. Under the same fixed law, $\mathbb{E}[S_H] = H\epsilon$. If $H\epsilon > L$, Hoeffding's inequality bounds $\Pr(S_H < L)$, while $S_H \ge L$ implies discovery during the episode. Including the trivial bound when $H\epsilon \le L$, the per-episode discovery probability is at least

$$\ell_\epsilon \coloneqq 1 - \exp\left(-\frac{[H\epsilon - L]_+^2}{2H}\right). \tag{25}$$

Count at most one discovery from each of the $n = \lfloor B/H \rfloor = 37$ full episodes, extending increment sequences beyond task completion if needed. These sequences are independent, and the learner retains every discovery. Thus m successful episodes imply task completion, while the learner may also acquire several events in one episode. The corresponding binomial tail gives

$$\Pr_{\rho_\epsilon}(\text{complete the task by } B) \ge R_\epsilon \coloneqq 1 - \sum_{j=0}^{m-1} \binom{n}{j} \ell_\epsilon^j (1 - \ell_\epsilon)^{n-j}. \tag{26}$$

This applies to either admissible order because their event laws are identical. For $\epsilon = 0.18$, $\ell_\epsilon \approx 0.2134$ and $R_\epsilon \approx 0.92032$. Thus an 18-percentage-point zero-to-forward transfer guarantees at least 92.0% success under the same model and budget, compared with the uniform upper bound 5.88×10^{-74}.

A.5 Intermediate Goals and Event Splitting

We retain the episode boundaries, action distribution, and total budget of the preceding model, and split each event into k successive goals of length $\ell = L/k$, with k a positive integer dividing L. Each discovered goal is recognized, retained, and replayed for free under the same idealization as a full event. The task now requires mk discoveries. There are no additional resets or local retry windows.

Table 1: Concentration bounds under zero-to-forward action priors in the main-text five-event model. Every entry is an upper or lower bound evaluated under the shared setting, not an exact success probability.

ϵ	Event upper bound q_ϵ	Task upper bound Q_ϵ	Task lower bound R_ϵ
0	5.18×10^{-17}	5.88×10^{-74}	0
0.04	3.92×10^{-10}	1.47×10^{-39}	0
0.075	1.18×10^{-5}	3.69×10^{-17}	0
0.1	0.00248	1.48×10^{-5}	0
0.12	0.0529	1	0
0.15	0.687	1	0
0.175	1	1	0.233
0.18	1	1	0.920

Theorem 3 (Event splitting under the same episode protocol)**.** *Under the common fixed increment law of the unit-step model, splitting each event into k retained goals does not decrease task completion probability within the budget B.*

Proof. Couple the split and unsplit learners using the same increment sequence in each episode, extending the sequence beyond task completion if needed. Let M_j be its running maximum in episode j, including the final partial episode. The respective discovery counts are

$$K_j^{(1)} = \left\lfloor \frac{M_j}{L} \right\rfloor, \qquad K_j^{(k)} = \left\lfloor \frac{kM_j}{L} \right\rfloor \geq kK_j^{(1)}.$$

Hence $\sum_j K_j^{(1)} \geq m$ implies $\sum_j K_j^{(k)} \geq mk$ on every coupled trajectory. Taking probabilities proves the claim. □

Within a single episode, splitting does not change the total net progress required for task completion. Its benefit comes from retaining intermediate discoveries across episode boundaries, reducing the progress lost when an unfinished event resets. The result assumes that intermediate outcomes can be recognized and reused, just as full events are in the main-text model.

Numerical evaluation of the bounds. For uniform actions, Equations 1 and 2 apply with event length $\ell = L/k$ and discovery count mk:

$$q_u^{(k)} = \exp\left(-\frac{L^2}{2k^2 H}\right), \qquad Q_u^{(k)} = \min\left\{1, 2[\min\{1, Nq_u^{(k)}\}]^{mk}\right\}. \tag{27}$$

Table 2 evaluates these same concentration bounds. Shorter goals reduce the magnitude of the single-event exponent, but the task upper bound becomes uninformative for $k \geq 5$. The non-decrease guarantee follows from Theorem 3, not from an increase in these upper bounds.

Table 2: Concentration upper bounds after event splitting under the same episode and retention protocol. Only goal granularity changes; the budget remains 100,000 interactions. A task upper bound of one is uninformative.

k	ℓ	mk	Event upper bound $q_u^{(k)}$	Task upper bound $Q_u^{(k)}$
1	450	5	5.18×10^{-17}	5.88×10^{-74}
3	150	15	0.0155	7.15×10^{-4}
5	90	25	0.223	1
10	45	50	0.687	1
30	15	150	0.959	1
90	5	450	0.995	1
450	1	2250	0.9998	1

Table 3: Functional components and interfaces of the upper-level program.

Component	Inputs	Outputs
Visual perception	Successive observations, previous perceptual state, and actual actions	Player and object estimates, scene geometry, observed events, and uncertainty.
Motion estimation and prediction	Perceived transitions, actual actions, and control context	Action-dependent movement estimates and predictions of relevant object trajectories.
Task and goal planning	Perceived state, motion predictions, task memory, and execution feedback	Current task, route, and local goal, with diagnostic planning summaries.
Outcome labeling	Causal perceptual readouts and recorded actions	Achieved outcomes and validity information under the shared goal–outcome semantics.

B Upper-Level Program Development

B.1 Program Architecture and Revision Workflow

Program components and interfaces. The framework specifies four functional responsibilities and their shared interfaces; the foundation model generates and revises their game-specific implementations. Table 3 summarizes the components.

High-level planning. High-level planning selects semantically meaningful tasks over a longer horizon and determines how to carry them out. For example, in Ms. Pac-Man, the program may choose to collect ordinary pellets, obtain a power pellet, or escape approaching ghosts. For tasks requiring movement, the planner selects a route and decomposes it into local goals. Across successive planning calls, the planner uses the current task and previously selected route as context. It evaluates whether to continue along the remaining route, revise the route in response to hazards or obstructions, or switch tasks as priorities and task conditions change. While following a retained route, the planner advances the local goal as observed progress confirms that a waypoint has been reached or passed. When the route or task changes, the planner selects a local goal consistent with the revised plan.

Development evidence and debug visualization. For each trial, the framework records returns, stopping conditions, gameplay frames, actual actions, goals, and diagnostic traces, linked to the tested program snapshot and executor checkpoint. It also generates debug videos that overlay the program's estimates of player and object locations, the current goal, and the planned route on the corresponding gameplay frames. These annotations are drawn from the program's perception and planning outputs. Comparing raw gameplay with the annotated video helps the development agent align code-level representations and decisions with observed game behavior and inspect discrepancies such as missed or mislocalized objects, inappropriate goal placement, and routes inconsistent with the visible scene geometry.

Developer–Auditor revision cycle. The Developer is the foundation-model agent responsible for program construction and revision. Program development proceeds in rounds of code revision, budgeted trials, independent auditing, and handoff. Within a round, the Developer can revise the program between trials, with each trial linked to its tested program snapshot and executor checkpoint. An audit is required after at most three finished trial attempts, including failed and comparison trials. A separate Auditor session examines gameplay before inspecting the code; the Developer's analyses and handoff notes remain hidden, while explicitly supplied previous audit issues serve as questions to recheck. The Auditor provides diagnostic findings and a round-end decision without editing the program or collecting additional interactions: `adopt` replaces the baseline (the currently confirmed program–executor pair) with the tested candidate; `rollback` retains the baseline; and `extend` retains the baseline while carrying the candidate forward for additional development trials in the next round. The decision identifies the corresponding tested program–executor pair and cites supporting trial evidence. After the audit, the Developer responds to the findings and writes a short handoff

recording supported capabilities, remaining issues, unresolved hypotheses, and relevant trial identifiers. The round then closes without further changes to the audited code. Subsequent revisions use the audit report and handoff together with the recorded development evidence.

B.2 Prompt Templates

The templates below are abridged versions of the implementation prompts. Bracketed fields are filled with game-specific knowledge or current runtime information. The shared context is combined with role-specific instructions; the Auditor receives information in two successive passes.

Shared context and program contract.

> **Context:** [Game and supplied rules]; [target score and action meanings]; [enabled goal fields]; [observation, program, and interaction-budget constraints].
>
> **Instructions:** Use supplied game knowledge and your own priors to reason about objectives, routes, hazards, timing, and recovery. Distinguish given rules, prior knowledge, observed evidence, and implemented capabilities. Calibrate visual recognition and action effects through interaction. Use permitted observations and feedback, without hidden emulator state. The program estimates visual state, selects persistent goals, and causally labels achieved outcomes. Follow the shared continuous/discrete goal format; leave learned primitive control to the neural executor.

The goal and labeling conventions are detailed in Appendix C.1. During initial exploration, additional instructions request controlled comparisons of holding versus briefly pressing an action, recording its effects, timing, context, and uncertainty.

Developer: construct, test, and revise the program.

> **Inputs:** [Shared context]; [current phase and remaining budget]; [candidate program and executor checkpoint]; [recent trial results]; [latest audit and short handoff]; [tool feedback].
>
> **Instructions:** Identify successful behavior, the main bottleneck, and one coherent improvement. Implement the change and request bounded online trials. Inspect gameplay frames before interpreting traces; distinguish program errors from a neural executor that is still learning. Distinguish initial exploration from subsequent trials with the neural executor, and link each subsequent trial to its tested program and executor checkpoint. Request independent review after testing. After the audit, reconcile its findings and update a short handoff for the next round. Record successful capabilities, resolved and unresolved issues, and supporting trial identifiers; close the round without further edits to the audited program.
>
> **Output:** Free-form analysis and code edits, followed by one JSON command on the last line: `continue`, `trial`, `review`, `finish_round`, or `blocked`. A trial request specifies its stopping condition, seed role, and purpose. The framework determines the step limit from the remaining development budget and the per-episode cap, charging only the steps actually executed.

Auditor, first pass: inspect gameplay independently.

> **Inputs:** [Shared context]; [gameplay frames]; [actual actions, rewards, and scores]. Candidate code and Developer interpretations are hidden.
>
> **Instructions:** Use game knowledge to describe normal progress. Identify successful, stalled, and failed behavior, possible missing capabilities, and anomalous mechanisms. Support observations with evidence and state uncertainty. Do not propose implementation changes or run trials.

Auditor, second pass: check the implementation and decide the next round.

> **Additional inputs:** [Current candidate code and paired checkpoint]; [confirmed baseline and its development evidence]; [full current-round trace]; [explicitly supplied previous audit issues, if any]. Developer analyses and handoffs remain hidden.
>
> **Instructions:** Check the first-pass concerns against the actual implementation. Report important mechanism bugs, missing mechanisms, and strategy bottlenecks supported by continuous trajectories, with code or gameplay evidence and behavioral consequences. Distinguish observed facts, code facts, hypotheses, and incomplete executor learning. Recheck previous issues rather than assuming they remain valid. Using development evidence only,

choose `adopt` to replace the baseline with the tested candidate, `rollback` to retain the baseline, or `extend` to carry the candidate into the next round for additional trials while keeping the baseline unchanged. Consider returns, seed coverage, trajectory completeness, and known failure modes; do not require monotonic score improvement. Return a concise report without prescribing code edits or running trials.

Output: Diagnostic findings; the `adopt`, `rollback`, or `extend` decision; the confirmed baseline pair identifier; any candidate pending additional trials; supporting trial identifiers; and the rationale and main uncertainties.

At the end of development, the last confirmed baseline is frozen for held-out evaluation. An `extend` decision does not promote a candidate without a subsequent `adopt` decision, and no separate Selector session is used. Held-out results are not used for revision or reselection. Runtime memory is initialized independently for each evaluation episode; development-derived knowledge in the program is frozen before evaluation.

C Lower-Level Learning and Joint Training

C.1 Shared Goal Encoding and Outcome Labels

The Atari implementation exposes seven fields: absolute target coordinates $(p_x^\star, p_y^\star)$, relative displacement (Δ_x, Δ_y), and three binary instructions $(b_{\text{fire}}, b_{\text{down}}, b_{\text{up}})$. The executor receives the vector $(p_x^\star/W, p_y^\star/H_{\text{img}}, \Delta_x/W, \Delta_y/H_{\text{img}}, b_{\text{fire}}, b_{\text{down}}, b_{\text{up}})$, with native image dimensions $W = 160$ and $H_{\text{img}} = 210$. Coordinates are real-valued inputs; the learner does not discretize them into goal bins. That is, the absolute and relative coordinates encode the same endpoint in complementary ways. We write $y^\star(g)$ for the target outcome corresponding to the endpoint and instruction bits requested by goal g. This target preserves their canonical meaning; state-dependent coordinate transformations below only change the input encoding.

The program's goal selector h chooses an endpoint and requested instruction bits. Achieved position comes from a causal programmatic visual readout; achieved instruction bits come from the actual primitive action's components, filtered by the game's enabled instructions. In particular, `up`/`down` label button components, not displacement signs, and `fire` does not label a successful hit. For enabled instructions, a false bit requests the absence of that component. Disabled bits impose no instruction constraint and are encoded as zero. They do not remove actions from the executor's action space.

Both online execution and replay use the same coordinate transformations. Within one coordinate chart, the relative displacement is target minus current position. Across known scene transitions or wraparound edges, observed entry/exit relations map the endpoint to the current chart. Scene identifiers and edge metadata compile these relations but are not additional actor inputs. Every replay candidate is re-encoded relative to the query state, and again relative to the successor state for the TD target. Unknown mappings are masked to avoid arbitrary cross-scene coordinate differences. The canonical target is unchanged by this state-dependent encoding.

Transitions require valid positions at both endpoints to enter the training pool. If online localization is unavailable, the runtime uses `FIRE` or `NOOP` according to the requested fire bit and excludes these fallback transitions from executor updates. With valid localization, the actor's selected action is sent directly to the environment without a code-based action replacement.

C.2 Population Fixed Point and Utility Interpretation

Population fixed point. Fix the program (including its labeler), policy π, and canonical goal semantics. Assume that x is sufficient for the dynamics and labels, and that replay covers the relevant state–action–goal queries while preserving $x' \mid x, a, g \sim P(\cdot \mid x, a)$. Assume that a common reference distribution μ dominates the relevant outcome distributions, that the critic can represent each Bellman target, and that population minimization is exact. We use density notation with respect to a common measure ν; integration includes discrete sums.

For a fixed normalized target critic $\widehat{d}_{\bar{\theta}}$, define

$$(\mathcal{T}^\pi \widehat{d}_{\bar{\theta}})(y \mid x, a, g) = (1-\gamma)\mathbb{E}_{x'}[K(y \mid x, a, x')] + \gamma \mathbb{E}_{x',a'}[\widehat{d}_{\bar{\theta}}(y \mid x', a', g)],$$

where $a' \sim \pi(\cdot \mid x', g)$. Since sampling $y \sim \mu$ with weight $\widehat{d}_{\bar{\theta}}/\mu$ integrates against $\widehat{d}_{\bar{\theta}}$, the conditional loss in Equation 9 is

$$-\int_{\mathcal{Y}} (\mathcal{T}^{\pi}\widehat{d}_{\bar{\theta}})(y \mid x, a, g) \log \widehat{d}_{\theta}(y \mid x, a, g)\, \mathrm{d}\nu(y).$$

This is cross entropy with the normalized Bellman target, so its realizable minimizer is $\widehat{d}_{\mathrm{new}} = \mathcal{T}^{\pi}\widehat{d}_{\bar{\theta}}$. The operator is a γ-contraction in the supremum of conditional total-variation distances: the immediate term cancels between two inputs, and averaging the remaining difference cannot increase that distance. Its unique fixed point is therefore d^{π} in Equation 6. This result concerns the population TD objective.

Utility interpretation. For an optional outcome utility $u(g, y)$, the occupancy induces the value

$$Q_u^{\pi}(x, a, g) \coloneqq \frac{1}{1-\gamma} \int_{\mathcal{Y}} u(g, y) d^{\pi}(y \mid x, a, g)\, \mathrm{d}\nu(y). \tag{28}$$

Integration here includes summation over discrete components. A scalar critic summarizes future outcomes for one utility; the occupancy model can support different queries over the represented outcomes. Note that this is an interpretation and optional extension, rather than a scalar reward used by the implemented learner. Environment returns guide the strategic revision loop.

For a discrete target outcome $y^{\star}(g)$ as defined in Appendix C.1, the utility $u(g, y) = (1-\gamma)\mathbf{1}\{y = y^{\star}(g)\}$ gives $Q_u^{\pi}(x, a, g) = d^{\pi}(y^{\star}(g) \mid x, a, g)$, the discounted occupancy mass of that outcome. For a continuous coordinate distribution with a density, an exact-point event has zero probability. A tolerance region or kernel would be needed to express approximate arrival as a scalar reward. The implemented actor instead matches a target among finite candidates; it does not interpret a point density or a candidate softmax probability as a calibrated task-success probability.

C.3 Finite-Batch Critic and Actor Updates

Replay and goal relabeling. A batch contains transitions (x_i, a_i, x_i') and immediate outcome labels y_i^{+}. Critic conditioning goals are drawn from other immediate outcomes using a random nonzero cyclic shift of the batch, then re-encoded at x_i and x_i'. A further shift supplies the bootstrapped candidate pool $\{y_j^{-}\}_{j=1}^{N}$. Thus, the two critic terms use the same empirical immediate-outcome marginal with different pairings. Samples with invalid coordinate mappings are masked.

Replay segments end at life loss or episode termination, and hindsight sampling stays within each segment. Stored next observations follow the recorded action and precede the next reset.

TD targets and loss. Each TD backup conditions the next action and target critic on the same canonical goal; only its relative-coordinate encoding changes with x'. It evaluates continued pursuit of that goal even when the data-collection policy later switches goals. The next action is sampled from the stochastic online actor, whereas development trials sample actions from a frozen execution policy π_{exec} (Appendix C.4). These off-policy backups evaluate the stochastic actor. The four-frame executor input approximates the sufficient state assumed in the population argument.

Suppressing these encoding maps and masks, sample $a_i' \sim \pi_{\omega}(\cdot \mid x_i', g_i)$ and define

$$\begin{aligned} F_{ij}^{\mathrm{next}} &\coloneqq f_{\theta}(x_i, a_i, g_i, y_j^{+}), & F_{ij}^{\mathrm{future}} &\coloneqq f_{\theta}(x_i, a_i, g_i, y_j^{-}), \\ \bar{F}_{ij} &\coloneqq f_{\bar{\theta}}(x_i', a_i', g_i, y_j^{-}), & \widetilde{W}_{ij} &\coloneqq \mathrm{stopgrad}[\mathrm{softmax}(\bar{F}_{i,:})_j]. \end{aligned} \tag{29}$$

For one critic head, the finite-batch approximation to Equation 9 is

$$\mathcal{L}_{\mathrm{TD}}^{(N)} \coloneqq (1-\gamma)\,\mathrm{CE}(F^{\mathrm{next}}, I_N) + \gamma\,\mathrm{CE}(F^{\mathrm{future}}, \widetilde{W}), \tag{30}$$

where $\mathrm{CE}(F, T) = -N^{-1}\sum_{i,j} T_{ij} \log \mathrm{softmax}(F_{i,:})_j$. Each target row sums to one. Equivalently, the density-ratio estimate $N\widetilde{W}$ is paired with a $1/N$ candidate average in the population loss. Terms involving only the reference measure do not affect critic gradients. The immediate and bootstrapped targets use observed labels and critic predictions, without environment rewards.

Twin critics and normalization. The implementation has two critic heads with normalized representation dot products as logits. Each online head minimizes its own cross entropy, and their

losses are averaged. For target weights and actor scores, each head is first normalized over candidates. The elementwise minimum of these log probabilities is then normalized again. Explicitly, $v_{ij}^{(q)} = \log \mathrm{softmax}(F_{i,:}^{(q)})_j$ and $s_{ij} = \log \mathrm{softmax}(\min_q v_{i,:}^{(q)})_j$. The target weights are $\mathrm{stopgrad}[\exp s_{ij}]$. This order removes head-specific row offsets before the minimum. Target critic parameters track the updated online critic using an exponential moving average, $\bar{\theta} \leftarrow (1-\eta)\bar{\theta} + \eta\theta$, with $\eta = 0.005$; the next action uses the online actor.

Discrete-instruction auxiliary objective. For each enabled instruction bit, the immediate-outcome logits also support a group contrastive loss: candidates with the same actual-action bit as the query form the positive set, and the loss is the negative log of their total softmax probability. Positives need not receive equal probability. The full critic loss is $\mathcal{L}_{\mathrm{critic}}^{(N)} = \mathcal{L}_{\mathrm{TD}}^{(N)} + 0.1\,\mathcal{L}_{\mathrm{instruction}}$, where the auxiliary term is averaged over enabled bits and both heads and omitted when no bits are enabled. This additional term is not part of the population fixed-point argument in Appendix C.2.

Batches cycle through enabled bits and sample 25% true and 75% false labels for the selected bit, falling back to uniform replay sampling if either group is unavailable. If more than one enabled bit is active in a sample, that sample is excluded from auxiliary queries and candidate sets, but remains in replay and the main TD loss. Auxiliary logits do not use spatial masks. The auxiliary loss uses recorded-action labels and leaves the legal action set unchanged.

Actor goals and candidate pools. For a sampled transition at time t, hindsight relabeling draws a valid future position at time $k > t$ within the same life/episode segment, with probability proportional to γ^{k-t-1}. The actor goal pairs that position with the instruction bits of the actual action a_t. The resulting batch supplies the actor's candidate pool, with the matching target on the diagonal. Relative coordinates for every candidate are recomputed for each query. The actor enumerates all legal actions and uses the normalized twin-critic score in Equation 10.

At the population fixed point, a critic represents $f^\star = \log(d^\pi/\mu) + b(x, a, g)$, with an arbitrary query-dependent offset b. Candidate normalization cancels this offset for each action, making scores comparable across actions within the same candidate pool. The actor uses hindsight targets, whereas the critic uses immediate outcomes, so their candidate marginals differ. The actor objective is consequently finite-candidate matching rather than exact log-density maximization.

C.4 Joint Upper- and Lower-Level Training

Algorithm 1 uses M and U for the frozen foundation model and current program, as in Section 4. During joint training, a *trial* is a bounded interaction period with a fixed program–executor pair. A *round* contains trials and intervening Developer revisions, ending with an independent Auditor review. The Developer–Auditor workflow is detailed in Appendix B.1. Audits provide diagnostic feedback and an `adopt`, `rollback`, or `extend` decision based on development evidence, without requiring monotonic score improvement.

Initialization and synchronization. During initial exploration, code-based control is used to test perception and action effects. The resulting trajectories are imported into replay. The executor then undergoes initial online training with a tested program held fixed. During subsequent rounds, the execution actor remains frozen within each round, and actions are sampled from its distribution. A background learner updates the online actor and critic from trial trajectories without running additional environments. Pending replay updates are completed before the updated actor is copied for the next round, or before development ends. Relabeling stored trajectories does not grant extra updates. The numerical schedule and network settings appear in Appendix D.2.

Program revision and replay consistency. A waypoint can persist across planning calls; its relative coordinates are updated without restarting the intended goal. Goal-replanning frequencies are specified in Appendix D.2.

The framework tracks a separate version of the outcome labeler and the components it depends on. When this version changes, the framework recomputes outcome labels for stored trajectories from recorded observations, actions, and goals in temporal order, using only information available at each transition, and updates the labels in replay. Changes to goal planning alone do not require relabeling if the labeler and its dependencies remain unchanged. Network and optimizer states are retained.

Algorithm 1 Joint training of PREINFORCE

Require: Frozen foundation model M, prior knowledge, development budget B, and update and audit schedules

Initialize an archive of tested program–executor pairs $\mathcal{V} \leftarrow \emptyset$ and development evidence $\mathcal{E}$
Initialize the confirmed pair $q_{\text{base}} \leftarrow \emptyset$
Construct and validate U using priors and initial code-based exploration to test perception and action effects
Initialize replay $\mathcal{D}$, actor π_ω, critic f_θ, and target critic parameters $\bar{\theta} \leftarrow \theta$
Import exploration trajectories into $\mathcal{D}$
Hold the tested program U fixed and train the executor online for 20,000 environment interactions
while development interactions remain **do**
    Begin a new round; complete pending replay updates
    Freeze the execution policy $\pi_{\text{exec}} \leftarrow \pi_\omega$
    while budget remains and an audit is neither due nor requested **do**
        Use M to propose U' using priors, U, q_{base}, and $\mathcal{E}$
        if the outcome labeler or its dependencies change **then**
            Causally relabel stored trajectories under U' and rebuild $\mathcal{D}$
            Retain actor, critic, and optimizer states
        end if
        $U \leftarrow U'$
        for each decision in the trial, while development budget remains **do**
            Update x_t, z_t; select or retain g_t and recompute its current encoding
            Sample $a_t \sim \pi_{\text{exec}}(\cdot \mid x_t, g_t)$, execute a_t, and observe r_t, o_{t+1}
            Update z_{t+1} and append the transition with $y_{t+1} = \kappa(z_t, a_t, z_{t+1})$ to $\mathcal{D}$
            for each scheduled replay update with valid data **do**
                Sample valid replay transitions, relabeled goals, and outcome candidates
                Update θ with TD InfoNCE and the discrete-instruction auxiliary loss (Appendix C.3)
                Update ω by goal–outcome matching (Equation 10)
                $\bar{\theta} \leftarrow (1-\eta)\bar{\theta} + \eta\theta$
            end for
        end for
        Record the trial's returns, trajectories, and diagnostics in $\mathcal{E}$
        If eligible, archive an immutable snapshot of (U, π_{exec}) in $\mathcal{V}$, linked to its development evidence
    end while
    Run an independent Auditor review
    On `adopt`, replace q_{base} with the adopted pair from $\mathcal{V}$; otherwise retain it
    On `extend`, carry the candidate forward for further trials; otherwise clear the pending candidate
    Record the audit report, decision, and any pending candidate in the Developer handoff and $\mathcal{E}$
end while
Complete pending learner updates and finalize $\mathcal{E}$
If $q_{\text{base}} = \emptyset$, terminate without evaluation
$(U^\star, \pi^\star) \leftarrow q_{\text{base}}$; freeze the confirmed pair
Evaluate the frozen pair under the protocol in Appendix D.1

The framework rejects invalid goals, undeclared instruction fields, and incompatible checkpoint or action formats.

D Experimental Setup and Additional Results

D.1 Environment, Development, and Evaluation Protocols

Environment and observations. We evaluate on the 26-game Atari 100k benchmark using `NoFrameskip-v4` environments. One interaction denotes one agent decision, with an action repeat of four, no sticky actions, and the minimal action set. The upper-level program receives the RGB frames from each action repeat, enlarged from 160×210 to 320×420 pixels by nearest-neighbor interpolation, together with unclipped rewards, step counts, and termination signals. For the neural executor, we take the pixelwise maximum of the last two frames in each action repeat, convert it to grayscale, resize it to 84×84 using bilinear interpolation, and stack four consecutive processed frames. Each reset executes NOOP for a uniformly sampled duration of 1–30 native frames, followed by one native frame of FIRE when available. No demonstrations, emulator RAM or internal state, or game-specific perception labels are provided.

Development budget and trials. The 100,000-interaction budget per game covers initial exploration, program trials, and executor training. For each trial, the foundation model can request a full episode or termination after a specified number of cumulative life losses, subject to the remaining budget.

Held-out evaluation. The selected pair remains frozen for one episode on each of five seeds unseen during development, under the fixed environment configuration. Evaluation trajectories and scores are used solely for reporting. The main results are based on one development run per game.

D.2 Model and Training Settings

Executor-training schedule. Initial executor training begins after a trial once cumulative development interactions reach or exceed 10,000, or earlier if requested by the foundation model. A tested upper-level program is then held fixed for 20,000 total interactions of initial online executor training, using eight parallel environments and rollouts of 32 decisions per environment. Subsequent development trials use frozen executor checkpoints, while a background learner consumes the newly recorded trajectories without running additional environments. Every 256 new transitions provide 64 minibatch updates, with a batch size of 256 and a replay capacity of 100,000. Updates require at least 2,000 valid replay transitions. Pending updates are completed before the next round is synchronized or development is closed. Replay relabeling does not grant extra update allowance.

Network architectures and optimization. The actor and critic use IMPALA-style image encoders on four stacked 84×84 grayscale frames, with channel widths $(64, 128, 128)$ and two residual blocks per stage. Both networks use layer normalization and MLP hidden layers of width 256; critic representations are 64-dimensional. We use Adam with actor and critic learning rates 3×10^{-5} and 3×10^{-4}, respectively, and global gradient norm clipping at 0.5. The TD and hindsight discounts are both 0.95. We decay the entropy coefficient linearly from 0.01 to 0.001 using a decay horizon of 100,000 steps.

Goal replanning. The program normally replans every four decisions, or earlier on goal completion, timeout, or task/scene changes. In Pong, it replans every decision.

D.3 Baselines and Per-Game Results

Baselines. We use the published per-game scores of BBF (Schwarzer et al., 2023) and EfficientZero V2 (Wang et al., 2024b). We reproduce Code policies (Weng, 2026) from its source code under our observation and interaction interfaces, providing the same human-supplied game knowledge as PREINFORCE. Both methods use `gpt-6-astra` with `high` reasoning effort. Code policies use the same 100,000-interaction development budget per game. During development, Codex maintains a best-so-far policy, which is frozen at the end and evaluated on five seeds unseen in development.

Per-game results and score sources. Table 4 reports per-game mean returns and aggregate human-normalized scores for seven RL baselines, Code policies (Weng, 2026), and PREINFORCE. Random and human reference scores follow Wang et al. (2016). The per-game RL reference in Figure 3 takes the highest published mean score among the seven RL baselines listed in the table: SR-SPR (D'Oro et al., 2023), EfficientZero (Ye et al., 2021), DreamerV3 (2025) (Hafner et al., 2025), STORM (Zhang et al., 2023b), DIAMOND (Alonso et al., 2024), BBF (Schwarzer et al., 2023), and EfficientZero V2 (Wang et al., 2024b). These general-purpose agents report results on all 26 games under a 100k online interaction budget. Published RL scores are taken from the comparison tables in Alonso et al. (2024); Wang et al. (2024b) and the supplementary material for DreamerV3 (2025), Table 4[1]. Each game's reference may come from a different agent. The figure uses point estimates.

Table 4: Per-game mean returns on Atari 100k. EZ and EZ-V2 denote EfficientZero and EfficientZero V2; Dreamer denotes the 2025 version of DreamerV3. Subscripts show SEMs over five evaluation seeds for PREINFORCE. Mean HNS and IQM use per-game mean HNS; IQM averages the middle 14 of the 26 sorted values. Bold and underlined entries indicate the best and second-best scores in each row, including the human reference.

Game	Human	SR-SPR	EZ	Dreamer	STORM	DIAMOND	BBF	EZ-V2	Code policies	PREINFORCE (Ours)
Alien	7127.7	1107.8	808.5	1118	983.6	744.1	1173.2	1557.7	2029.3	**8414.0**$_{\pm 1085.4}$
Amidar	1719.5	203.4	148.6	97	204.8	225.8	244.6	184.9	485.3	**6201.2**$_{\pm 1368.1}$
Assault	742.0	1088.9	1263.1	683	801.0	1526.4	2098.5	1757.5	1968.7	**6296.2**$_{\pm 1517.1}$
Asterix	8503.3	903.1	25557.8	1062	1028.0	3698.5	3946.1	61810.0	253563.3	**999500.0**$_{\pm 0.0}$
Bank Heist	753.1	531.7	351.0	398	641.2	19.7	732.9	**1316.7**	627.3	916.0$_{\pm 68.5}$
Battle Zone	37187.5	17671.0	13871.2	20300	13540.0	4702.0	24459.8	14433.3	15066.7	**58400.0**$_{\pm 7698.1}$
Boxing	12.1	45.8	52.7	82	79.7	**86.9**	85.8	75.0	27.7	32.0$_{\pm 8.1}$
Breakout	30.5	25.5	414.1	10	15.9	132.5	370.6	400.1	397.1	**439.6**$_{\pm 82.8}$
Chopper Command	7387.8	2362.1	1117.3	2222	1888.0	1369.8	7549.3	1196.6	20300.0	**38120.0**$_{\pm 5806.9}$
Crazy Climber	35829.4	45544.1	83940.2	86225	66776.0	99167.8	58431.8	**112363.3**	33473.3	54140.0$_{\pm 4060.7}$
Demon Attack	1971.0	2814.4	13003.9	577	164.6	288.1	13341.4	22773.5	**33332.0**	8290.0$_{\pm 747.4}$
Freeway	29.6	25.4	21.8	0	33.5	33.3	25.5	0.0	31.6	**34.0**$_{\pm 0.0}$
Frostbite	4334.7	2584.8	296.3	3377	1316.0	274.1	2384.8	1136.3	5008.7	**9656.0**$_{\pm 815.6}$
Gopher	2412.5	712.4	3260.3	2160	8239.6	5897.9	1331.2	3868.7	7889.3	**64448.0**$_{\pm 1130.3}$
H.E.R.O.	30826.4	8524.0	9315.9	13354	11044.3	5621.8	7818.6	9705.0	18915.0	**32503.0**$_{\pm 141.9}$
James Bond	302.8	389.1	517.0	540	509.0	427.4	1129.6	468.3	603.3	**4270.0**$_{\pm 2506.9}$
Kangaroo	3035.0	3631.7	724.1	2643	4208.0	5382.2	6614.7	1886.7	4586.7	**6700.0**$_{\pm 905.0}$
Krull	2665.5	5911.8	5663.3	8171	8412.6	8610.1	8223.4	**9080.0**	4700.0	2856.0$_{\pm 274.4}$
Kung Fu Master	22736.3	18649.4	30944.8	25900	26182.0	18713.6	18991.7	28883.3	14573.3	**43400.0**$_{\pm 8535.2}$
Ms. Pac-Man	6951.6	1574.1	1281.2	1521	2673.5	1958.2	2008.3	2251.0	5058.0	**8504.4**$_{\pm 2339.9}$
Pong	14.6	2.9	20.1	-4	11.3	20.4	16.7	**20.8**	9.5	18.0$_{\pm 0.9}$
Private Eye	69571.3	97.9	96.7	3238	7781.0	114.3	40.5	99.8	23337.1	**99610.4**$_{\pm 96.3}$
Q*bert	13455.0	4044.1	13781.9	2921	4522.5	4499.3	4447.1	16058.3	**33100.0**	32680.0$_{\pm 794.9}$
Road Runner	7845.0	13463.4	17751.3	19230	17564.0	20673.2	**33426.8**	27516.7	16720.0	24640.0$_{\pm 3460.4}$
Seaquest	42054.7	819.0	1100.2	962	525.2	551.2	1232.5	1974.0	19566.0	**179334.0**$_{\pm 13395.0}$
Up N Down	11693.2	**112450.3**	17264.2	46910	7985.0	3856.3	12101.7	15224.3	7074.7	11952.0$_{\pm 3281.2}$
Mean HNS	1.000	1.271	1.943	1.247	1.266	1.459	2.247	2.680	3.649	**9.202**
IQM	1.000	0.700	1.047	0.543	0.679	0.757	1.139	1.305	1.443	**2.445**

D.4 DEVELOPMENT DYNAMICS ON ALIEN

As an example of how our upper-level policy and lower-level policy evolve, Figure 6 illustrates how evaluation score and goal-reaching rate evolve during development on Alien. Goal-reaching rate measures the fraction of issued goals for which the player comes within two native pixels of the target before the goal is replaced or its execution ends. Both metrics improve during initial fixed-program training. Subsequent joint development achieves higher returns and goal-reaching rates, although returns fluctuate across revisions. These later changes reflect the evolving executor, upper-level goals, and perceptual readout.

D.5 ABLATION PROTOCOL

Experimental design. We select eight games whose per-game RL reference scores fall below the corresponding human reference scores. For each game, the reference code is the final strategy obtained through PREINFORCE. We construct three auxiliary-reward variants from this same final strategy source using `gpt-6-astra`, with `high` reasoning effort, and train each variant with three

[1] https://www.nature.com/articles/s41586-025-08744-2

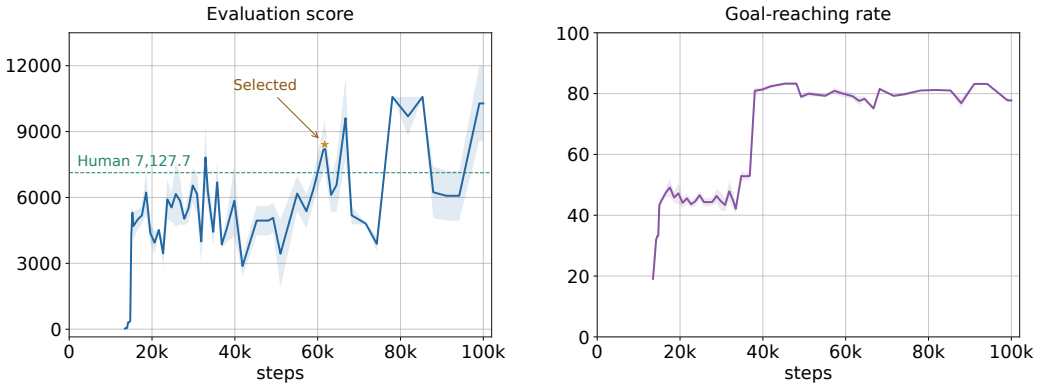

Figure 6: Development dynamics on Alien. **Left:** evaluation score. **Right:** goal-reaching rate within two native pixels (%). Curves show means, with shaded bands indicating ± SEM over five evaluation seeds. The dashed line marks human performance, and the star marks the selected program–executor pair.

BBF training seeds, $\{1, 2, 3\}$, yielding nine runs per game. After program selection, each run initializes a fresh BBF agent and trains it for 100,000 additional agent–environment interactions, with the selected program and auxiliary-reward implementation held fixed. The agent receives standard image observations and scalar rewards. We report the average results in Figure 4.

Auxiliary-reward construction. The reward variants are developed in separate authoring runs. A condensed description of the reward-generation instruction is:

> Using the supplied strategy source code, implement a game-specific auxiliary reward that encourages progress toward the game's objectives. Use observable transition evidence, limit repeated rewards for the same event, and handle episode resets and life loss. Preserve the agent's observations, actions, and environment dynamics. Optimize the implementation's runtime without changing its reward semantics.

The reward computation receives RGB frames associated with the transition, the executed action, the original environment reward, the remaining number of lives, and a terminal indicator. It may maintain internal state to track progress and previously rewarded events.

Let r_t denote the original environment reward and r_t^{aux} the sum of the weighted auxiliary terms. The reward used for learning is

$$\tilde{r}_t = \text{clip}(r_t, -1, 1) + \text{clip}(r_t^{\text{aux}}, -0.2, 0.2).$$

The auxiliary coefficients are fixed within each variant and shared across its training seeds; no additional clipping is applied to the combined reward. Evaluation disables auxiliary rewards and uses undiscounted, unclipped environment returns.

BBF training and evaluation. We use the BBF implementation and base configuration, with a replay ratio of eight gradient updates per environment interaction, a batch size of 32, and a network-reset interval of 5,000 interactions. Other learning settings, including the network architecture, optimizer, replay mechanism, and observation preprocessing, follow the BBF base configuration. Each final model is evaluated for five episodes with an evaluation exploration rate of $\epsilon = 0.001$. For Frostbite, an episode is additionally truncated after 2,000 consecutive interactions with zero original environment reward to handle the observed non-terminating game state. Auxiliary rewards do not reset this counter.

D.6 Computational Cost

We summarize resource usage for eight games: Alien, Amidar, Battle Zone, Freeway, Frostbite, Ms. Pac-Man, Pong, and Private Eye. Across eight measured games, development uses an average of

6.01M uncached input tokens, 83.46M cached input tokens, and 0.581M output tokens per game. Development includes program construction, auditing, environment trials, and executor training. Active development time averages 16.19 hours per game.