# LEARNING STATEFUL PREDICTIVE KNOWLEDGE FROM EXPERIENCE

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

## ABSTRACT

As large language model (LLM) agents increasingly learn from experience, they primarily rely on trajectory-level reflection to extract insights. Viewed through the lens of predictive knowledge (Sutton et al., 2011), we argue that this approach operates on episodic hindsight rather than predictive foresight, yielding brittle, path-dependent heuristics. To address this, we propose **Stateful Knowledge Learning** (SKL). SKL shifts the agent's focus from trajectory-level summarization to maintaining Stateful Knowledge: explicit, declarative predictive assessments anchored to state. We first demonstrate a motivating example showing how stateful knowledge provides granularity, enhances generalization, and enables knowledge bootstrapping. To further scale up the idea, we introduce two algorithms via self-distillation (SKL-SD) and reinforcement learning (SKL-RL), training agents to autonomously extract state-grounded predictive knowledge from experience and learn to leverage it for policy making. Experiments on WebShop, ScienceWorld, and more complex task ChessPuzzles show that SKL provides a scalable way to organize and internalize reflective computation, yielding performance gains under the stated experience and simulation budgets. The code is available at `https://anonymous.4open.science/r/Stateful_Knowledge_Learning-4265/`.

All knowledge about the world is predictive. — *Richard Sutton*

## 1 INTRODUCTION

The frontier of artificial intelligence is increasingly shifting towards the "Era of Experience" where agents acquire capabilities and knowledge through continuous, grounded interactions with their environments rather than relying solely on static human data (Silver & Sutton, 2025). Recent efforts have made significant progress for large language model agents, including Reflexion (Shinn et al., 2023), advanced memory systems (Zhou et al., 2025), and more recent learning-based methods (Zhang et al., 2025b; Shi et al., 2026b). These frameworks typically rely on the idea of self-reflection: analyzing past interaction trajectories to summarize errors and extract insights, which can subsequently guide the agent's next cycle of rollout.

This growing reliance on experiential learning raises a fundamental question: **what exactly constitutes "knowledge" for an LLM agent to learn from experience?** While "knowledge" is a broad concept with numerous definitions (Zagzebski, 2017) and instances (static facts or mathematical logic), in the context of agent interacting with dynamic environments, we believe **predictive knowledge** offers a highly principled framework (Sutton et al., 2011; Littman & Sutton, 2001) to capture the core of agent environment knowledge: Much of an agent's knowledge about the world is intrinsically predictive, meaning it can be translated into statements and predictions about potential future outcomes. For an LLM agent, this means expressing forward-looking assessments, whether it is estimating the exact numerical distance to a sub-goal, deducing hidden transition dynamics governing the environment, or predicting whether a strategic opportunity exists in the following steps.

Viewed through this lens of predictive knowledge, the dominant trajectory-level reflection paradigms reveal a structural limitation: they operate on episodic hindsight rather than predictive foresight. A fundamental premise of predictive knowledge is that predictions about the future must be state-grounded, not trajectory-grounded (Sutton et al., 2011). The future unfolds based on the current



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.

state[1], regardless of the historical path taken to reach it. When agents evaluate entire trajectories post-hoc, they can conflate environment dynamics with the particular historical sequence that was sampled. Consequently, they may yield path-dependent heuristics (e.g., "I should avoid moving right early") instead of state-grounded predictions (e.g., "Moving right is hazardous when a pit is adjacent"). To genuinely harness predictive knowledge, we shift the update target of agent reflection from trajectory summaries to **Stateful Knowledge**: explicit, declarative predictive assessments maintained for encountered states. Here, "stateful" refers to indexing, updating, and retrieving the assessment by the current task-provided state representation.

In this paper, we propose **Stateful Knowledge Learning (SKL)**, which recasts self-reflection as an iterative, state-indexed, value-iteration-inspired process for refining natural-language predictive knowledge from experience, which we name it *knowledge bootstrapping*. The paper is organized as follows:

**Understanding Stateful Knowledge.** Through a motivating experiment on a stochastic FrozenLake environment, Section 2 makes the idea visible with fixed model weights. It studies how state-grounded assessments can provide finer-grained predictive guidance and how knowledge bootstrapping propagates such assessments backward from successor states.

**Scaling and Internalizing Stateful Knowledge.** Section 3 presents SKL-SD and SKL-RL as two ways to scale the same knowledge bootstrapping principle. SKL-SD reuses state-matched evidence from a replay buffer and internalizes verified updates through self-distillation, whereas SKL-RL gathers state-matched evidence online through parallel simulation and jointly trains the simulation–bootstrapping loop with reinforcement learning. Experiments in Section 4 study these scaling paths on WebShop, ScienceWorld, and ChessPuzzles, together with related ablation studies. Our claim is not universal or token-efficient superiority over reflection baselines; rather, SKL provides a **scalable way to organize, propagate, and internalize** self-reflection under matched experience instances or stated simulation budgets.

## 2 Understanding stateful predictive knowledge: a motivating example

We define **Stateful Predictive Knowledge** $z^\pi$ as a policy-dependent, language-valued mapping from state $s \in \mathcal{S}$ to a language assessment:

$$z^\pi : \mathcal{S} \to \mathcal{L}_{\text{lang}}, \qquad s_t \mapsto z^\pi_{s_t}. \tag{1}$$

The assessment $z^\pi_{s_t}$ is (1) *state-indexed*: the agent is asked to reason about and update only the designated current state $s_t$, and the resulting knowledge is stored and retrieved using $s_t$ as its anchor; (2) *predictive*: the agent observes possible downstream outcomes through experiences aggregated from $s_t$ and updates its assessment of what the state implies for the future; and (3) *decision-relevant*: the updated assessment directly conditions the next action, $a_t \sim \pi(\cdot \mid s_t, z^\pi_{s_t})$. More details about how $z$ is updated analogous to MC/TD update can be found in Appendix C.

### 2.1 Reflect on trajectories versus on stateful knowledge

We provide a motivating example to demonstrate how different methods extract information under the same interaction budget. We distinguish our state-grounded approach from standard reflection paradigms as follows (also as illustrated in Figure 6):

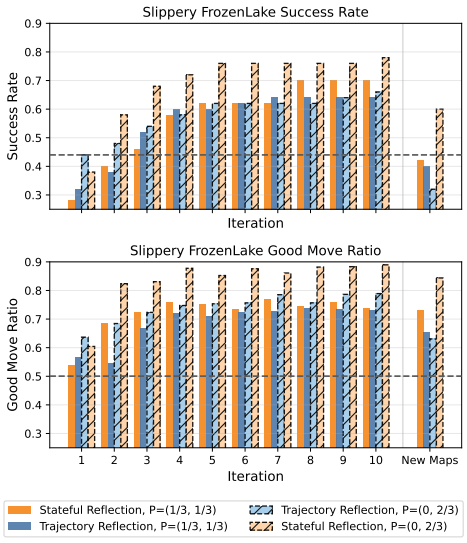
**Trajectory Reflection (Hindsight):** At iteration $i$, guided by a plan from the previous trial, the agent $\pi$ interacts with environment $E$ to form a complete trajectory $\tau(s_0) = (s_0, a_0, s_1, ..., s_T)$ starting at initial game state $s_0$. The agent then reflects on the entirety of $\tau$ to generate a global critique and an updated previous plan for iteration $i + 1$ (e.g., *Reflexion* (Shinn et al., 2023)). The process repeats until the maximum number of iterations $I$. In Appendix D, we also implement a variant where the agent reads multiple previous trajectories to update plans.

[1]Here, *state* denotes the observable environmental state, which assumes different representations depending on the specific task: a textual grid in FrozenLake, a parsed webpage in WebShop, an observation in ScienceWorld, or Forsyth-Edwards Notation (FEN) in ChessPuzzles.

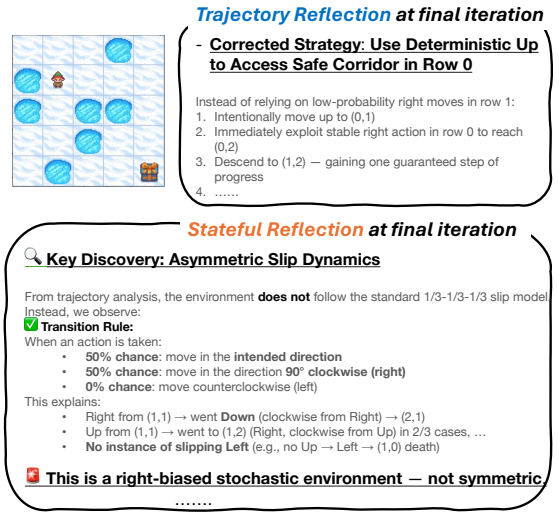
**Stateful Knowledge Reflection (Foresight):** Throughout the iterations, the agent $\pi$ consistently maintains an explicit, state-indexed predictive knowledge table $\{z_s\}_{s\in\mathcal{S}}$ for each visited game state $s \in \mathcal{S}$ by querying: *"What does the current state predict about future outcomes?"*. At each iteration, decisions $a$ are conditioned directly on this declarative assessment $z_s$, i.e., $a \sim \pi(\cdot|z_s, s)$, to interact with the environment $E$. During the reflection phase, rather than summarizing from a single trajectory, the agent first aggregates outcomes associated with a specific state $s$ across multiple previous trials, denoted as $\{\tau_n^H(s)\}_{n=1}^N$ at budget $(N, H)$. $N$ is the number of considered trials and $H$ is the maximum horizon. Then the agent generates a refined knowledge $\hat{z}_s \sim \pi(\cdot|\{\tau_n^H(s)\}_{n=1}^N, s, z_s)$ based on the contexts, updates the knowledge table and proceeds to the next iteration. Refer to Figure 6 for a more detailed illustration.

To understand how stateful knowledge works, we utilize a stochastic variant of the FrozenLake environment (Feng et al., 2024), where the agent must maneuver toward a goal tile while avoiding deadly pits on the icy surface. The game state $s \in \mathcal{S}$ contains a textual grid representing the spatial location of elements in the task. Crucially, the "icy" surface introduces transition stochasticity: an intended directional movement (e.g., "Up") has a probability of slipping into orthogonal directions (e.g., "Left" or "Right"). While many recent LLM-agent studies focus on deterministic transitions, this slipping mechanic introduces an extra layer of complexity that perfectly exposes the myopia and brittleness of trajectory-grounded reasoning, since an optimal decision might fail by chance, while a fatal blunder might luckily succeed. Agents relying on trajectory hindsight are easily misled.

We prompt the strong model *Qwen3-235B-A22B* (Yang et al., 2025). To prevent the agent from cheating by pre-trained knowledge about FrozenLake (the default is $P = (1/3, 1/3)$, so when move up, $P(\text{"up"}|\text{"up"}) = P(\text{"left"}|\text{"up"}) = P(\text{"right"}|\text{"up"}) = \frac{1}{3}$), we also test a variant by setting the slipping probability to $P = (0, 2/3)$. To maintain the same reflection budget with the trajectory baseline, the stateful agent only records and updates the initial state $z_{s_0}$ in the knowledge table. Knowledge for the remaining states is regenerated, taking the new root-state knowledge into account. For metrics, we evaluate the agent's success rate of reaching the goal and good move ratio (the probability of an action that yields the lowest death rate). We use 50 maps for training and the remaining 50 as the held-out test set.




(a) Iterative quantitative performance.




(b) Qualitative examples

Figure 1: **Slippery FrozenLake Analysis.** (**a**) Comparison of trajectory-level and stateful knowledge reflection. The grey dashed line represents the baseline for retrying the game ten times without any self-reflection and success once. Good move ratio refers to the probability of an action that yields the lowest death rate. (**b**) Representative examples of agent behaviour in different iterative loops. In Appendix D, we also include a complementary experiment where trajectory reflection reads multiple past trajectories and still fail to generalize.

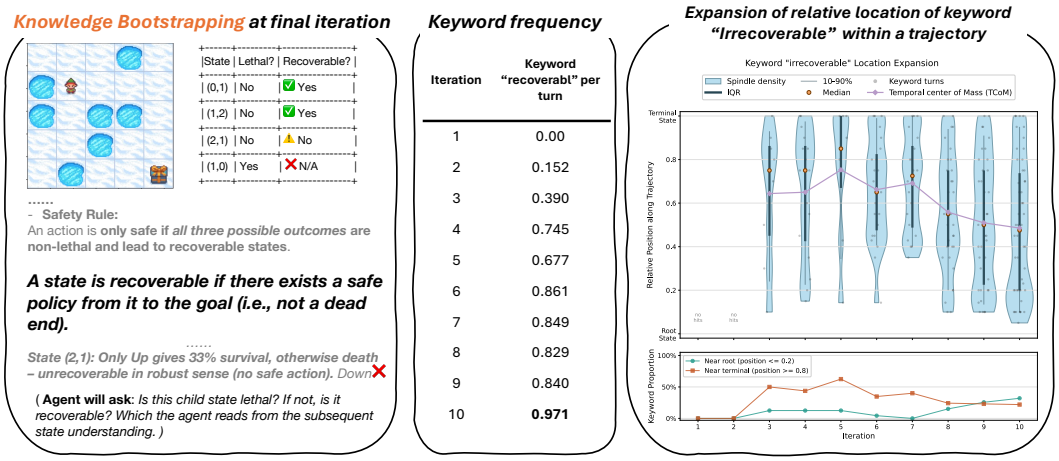



Figure 2: **Left**: An example reasoning trace from the fully bootstrapping variant at the final iteration. The agent defines and utilizes *recoverability* for reasoning. **Middle**: The average frequency of the keyword *recoverability* across training iterations, showing a steady upward trend. **Right**: The upper panel shows the distribution of the relative trajectory position at which the target keyword appears across iterations. The lavender curve reports the mean occurrence position along the trajectory, where a downward shift indicates the keyword occurrences (*irrecoverable*) are expanding backward from terminal states toward earlier/rootward states. The lower panel summarises this redistribution by plotting the fraction of keyword samples appearing near the root and near the terminal state. Together, the two panels show a transition from sparse terminal-side occurrences to broader coverage across earlier timesteps.

We report both performance among iterations on training environments and the final test performance (after iteration 10) in Figure 1, which provides evidence for our predictive knowledge hypothesis:

**Robustness to Stochasticity (Overcoming Hindsight Bias):** State-level reflection achieves the best observed performance in our setting. As shown in the qualitative example in Figure 1b, trajectory critiques suffer from severe hindsight bias: they prematurely conclude that certain movements are deterministic after a single lucky success, leading to brittle policies. Conversely, stateful reflection successfully predicts and uncovers the asymmetric slipping dynamics.

**Generalization and Transfer (State vs. Path):** When transferred to a newly generated testing map after 10 iterations, all agents experience a performance drop. However, on the tweaked scenario, trajectory-level reflection shows a drastic drop (-33%) while stateful knowledge drop is mild (-16%). This indicates that, in this setting, state-specific predictive insights (e.g., "slipping only occurs to the right from this type of tile") can be more generalizable than the trajectory-based baseline.

### 2.2 KNOWLEDGE BOOTSTRAPPING

Another benefit of stateful knowledge is **Knowledge Bootstrapping**. This key idea shares similarity with temporal difference learning (TD) (Sutton et al., 1998): because of the temporal relation between states (e.g., $s_t, a_t \rightarrow s_{t+1}$), an agent can update its current assessment on $s_t$ based on the knowledge of successor states $s_{t+1}$. Maintaining stateful knowledge makes successor-to-predecessor propagation an explicit update operation, whereas such temporal dependencies are not explicitly represented in standard trajectory-level reflection.

To demonstrate this, we deploy a *Fully Bootstrapping Agent* in the FrozenLake environment, where the knowledge of all visited states in a trajectory is updated. The aim is to show how knowledge can propagate across all visited states in the trajectory (workflow also illustrated in Figure 6). Now the agent updates recursively backwards from the terminal state $s_T$ to the root state $s_0$, thinking based on the concatenation of context:

$$[s_t, \{\tau_n^H\}_{n=1}^N, z_{s_{t+H}}, z_{s_t}] \tag{2}$$

The context includes the current state, aggregated local experience over horizon $H$ and the pre-existing stateful knowledge of the destination state $s_{t+H}$ and current state. This setup encourages

the agent to comprehend its own self-generated intermediate assessments and adjust its current understanding accordingly, achieving temporal consistency of knowledge. For instance, if an action leads to a successor state $z_{s_{t+H}}$ that the agent already recognizes as high-risk, it must update $z_{s_t}$ to reflect this derived danger, effectively "thinking ahead" by $H$ steps. Figure 2 illustrates the performance of the fully bootstrapping agent with a look-ahead horizon $H = 1$, providing a clear visualization of how predictive knowledge propagates temporally across states. This analogy to temporal-difference learning is conceptual: natural-language synthesis does not inherit Bellman contraction or convergence guarantees, and incorrect successor assessments may propagate backward.

**Emergent Semantic Bootstrapping.** The fully bootstrapping agent reveals an emergent semantic bootstrapping behaviour. We show an example of such emergence in Figure 2, tracing how the agent develops a strategic concept called *recoverability* during train-time, which is an emergent concept defined by the agent itself, denoting if there exists a safe policy from the state to the goal. As the agent iterates, the keyword "recoverable" (and its variants) occurs with increasing frequency, indicating the consolidation of this abstract concept. Crucially, as also observed in Figure 2, the relative position of the assertive keyword *irrecoverable* starts from terminal states, then gradually shifts back toward the root state. This trend is consistent with the agent discovering critical bottlenecks and propagating consistently that information backward through its natural language knowledge base, allowing it to preemptively identify fatal sequences long before they conclude. Appendix L.1 also reports corruption and repeated-error analyses.

## 3 Training LLM agents to extract and learn from stateful knowledge

We are now encouraged to scale Stateful Knowledge Learning on more complex tasks, such as with more complex state representation or longer horizons. Since current off-the-shelf models are typically not trained to be state-aware (Shojaee et al., 2025), we aim to internalize this capability into the model's parameters. Revealed by previous experiments, the key to a Stateful Knowledge Learning (SKL) agent is a system that learns to iteratively refine its stateful knowledge $z$ by repeating the following two steps:

**1. Extracting stateful knowledge from experience**: Aggregates state-specific outcomes across multiple visits (with budget $N, H$): $\{\tau_n^H(s_t)\}_{n=1}^N = \{(s_t, z_{s_t}, a_t^{(n)}, s_{t+1}^{(n)}, ..., s_{t+H}^{(n)})\}_{n=1}^N$ as informative contexts. Higher budgets increase the **representational power**. A bootstrapping loop then extracts new knowledge $\hat{z}_{s_t}$ based on the aggregated experience as well as the evaluative signals of its successor state $z_{s_{t+H}}$, acting as a **temporal regularizer** that ensures assessments are grounded in the agent's own predictive logic. We follow the idea of knowledge bootstrapping (Section 2.2) for the remaining parts of this paper (by setting $H < T$).

**2. Internalize stateful knowledge to guide policy**: The extracted knowledge is internalized and leveraged to guide policy optimization. While Section 2 utilizes stateful knowledge as a plug-in context for inference, we propose more scalable variants to inject stateful knowledge into model parameters.

We introduce two primary training variants that perform steps 1 and 2 differently: **SKL Self-Distillation (SKL-SD)** and **SKL Reinforcement Learning (SKL-RL)**, see Figure 8 for visualization. The core distinction lies in how they manage to balance between the representation and regularization (more detailed discussion in Appendix E): SKL-SD samples model-free rollouts from replay buffer, whereas SKL-RL employs online, model(simulator)-based search within a single episode. Pseudocode is provided in Appendix A. Detailed description of SKL-SD and SKL-RL is in Appendix F. A comparison with contemporary reflection-based methods is available in Appendix G.

**Intended use.** SKL-SD is the more practical plug-in variant for current post-training or self-distillation pipelines, with configurable additional state-update cost. SKL-RL is a more autonomous but substantially more expensive prototype that requires a resettable, state-settable simulator; here it is used to study joint optimization of simulation and semantic bootstrapping rather than presented as the more practical deployment choice.

# 4 EXPERIMENTS

We first evaluate SKL-SD on established agentic benchmarks (WebShop and ScienceWorld) to demonstrate that stateful knowledge learning can be integrated into existing training frameworks and compared with current reflection training methods. Subsequently, we explore SKL-RL on ChessPuzzles, a complex reasoning task that suffers less from data contamination and requires deep look-ahead and strategic state assessments.

**Evaluation protocols.** These experiments answer different questions and should not be read as one common evaluation setting. In FrozenLake, model weights remain fixed while an external plan or state-knowledge table is updated across ten trials; the final memory is then frozen for one-shot transfer to held-out maps. SKL-SD instead updates model weights during post-training and is evaluated directly on held-out WebShop and ScienceWorld tasks without test-time reflection or retry. SKL-RL is evaluated with the stated simulator-assisted budget $(N, H)$; the distillation ablation uses the same simulator-assisted protocol, while a separate zero-simulation evaluation is reported below as a negative result.

## 4.1 SKL-SD ON AGENTIC ENVIRONMENTS

**Experiment Setup.** We compare SKL-SD to other training baselines investigated in (Shi et al., 2026b) and follow their exact experiment setups. While SKL-SD shares the feedback-learning DNA of these methods (Appendix G), it replaces trajectory-level reflection with state-grounded predictive knowledge bootstrapping. Detailed setup is in Appendix J.1. The bootstrapping parameters are set to a horizon $H = 3$ and a state-aggregation budget $N = 3$. Sampling from the buffer $\mathcal{B}$ follows a recency-prioritized ordering, where samples are ranked according to their freshness and the latest experiences are selected first (Ma et al., 2026). *Qwen2.5-7B-Instruct* is adopted as the backbone model.

**Results** Performance is reported in Table 1. On Webshop, SKL-SD reaches 0.790, improving over the strongest baseline R$^3$L (0.757) by +3.3 absolute points, and outpacing standard GRPO (0.709) as well as other reflection-based methods such as Reflect-GRPO (0.723) and Critique-GRPO (0.714). In the more difficult ScienceWorld environment, SKL-SD again achieves the best result (0.422 vs. 0.403 for R$^3$L and 0.388 for Critique-GRPO). These entries are reported point estimates; without repeated-seed uncertainty estimates, we do not claim statistical significance for the smaller differences.

These gains demonstrate that **replacing trajectory-level reflection with stateful knowledge learning can potentially bring benefits to larger-scale tasks**. A review of the reasoning traces (see Appendix J.2) reveals that the SKL-SD agent develops a structured, hierarchical understanding of the task. For example in WebShop, at the initial search stage, the agent generates precise notes on required attributes and formulates a contingency plan for potential search failures—a behaviour rarely seen in the other tested trajectory-level training baselines. During the purchasing phase, the predictive knowledge $z$ shifts to specific product verification, demonstrating that stateful learning helps the agent maintain a "mental map" of the task progress, leading to more grounded decision-making.

| **Methods** | **WebS.** | **SciW.** |
|---|---|---|
| RAFT | 0.682 | 0.201 |
| OPMD | 0.684 | 0.359 |
| GRPO | 0.709 | 0.378 |
| GSPO | 0.720 | 0.363 |
| Reflect-GRPO | 0.723 | 0.356 |
| Critique-GRPO | 0.714 | 0.388 |
| R$^3$L | 0.757 | 0.403 |
| **SKL-SD** (ours) | **0.790** | **0.422** |

Table 1: Performance comparison on WebShop (WebS.) and ScienceWorld (SciW.). Baseline performance borrowed from (Shi et al., 2026b)

**Practical Consideration** While SKL-SD introduces additional computational overhead, primarily due to longer model outputs during stateful rollouts and context prefilling for bootstrapping, we bootstrap on only $20\%$ of states uniformly sampled across a trajectory, limiting the increase in training time to approximately $1.5\times$ compared with the reflection baselines. Appendix M reports the state-coverage sweep and token-aligned results.

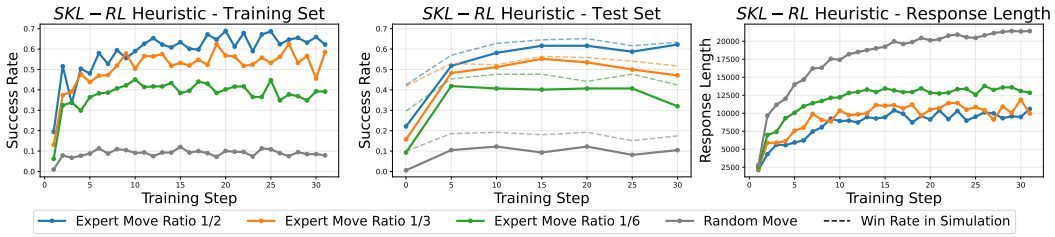



Figure 3: Experiments of SKL-RL with heuristic search simulation on ChessPuzzles.

### 4.2 SKL-RL ON CHESSPUZZLES

We next evaluate how SKL-RL can operate in a more autonomous way without complex data orchestration. We test on ChessPuzzles (Ruoss et al., 2024), a domain where even the strongest LLMs fail due to incomplete internalized chess knowledge (Liu et al., 2025; Hwang et al., 2025). This limited prior knowledge setting is especially diagnostic for our framework, as it directly tests the agent's ability to learn from experience. We elaborate on our motivation in Appendix K.1 and detail the experimental setup in Appendix K.2.

In the following sections, we evaluate $\mathcal{L}_{GRPO}(\mathcal{D}_{\text{SKL-RL}})$ across different parallel simulation strategies in Sections 4.2.1 and 4.2.2, and ablate the distillation loss in Section 4.2.3.

#### 4.2.1 EXPERIMENTS WITH HEURISTIC PARALLEL SIMULATION

To isolate whether the bootstrapping loop can be learned from outcome signals alone, we control simulation quality using *Stockfish* chess engine (Romstad et al., 2024). We construct behaviour policies by mixing expert moves with random sub-optimal moves at expert probabilities of $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{1}{6}$. Figure 3 shows that higher expert rates yield faster learning and stronger asymptotic performance. The agent progressively learns to identify winning positions during simulation and steer toward them, with win probability in simulated trajectories rising accordingly. Response length correlates positively with the difficulty of detecting expert moves, indicating that noisier simulation drives the agent to invest more internal reasoning to extract state-grounded predictions. Detailed traces (Appendix K.3) further reveal that under high signal (i.e. expert move $1/2$), knowledge bootstrapping converges rapidly to accurate but shallow assessments; whereas under low signal (i.e. random), it remains structured but increasingly prone to hallucination. This is a direct manifestation of the exploration–exploitation trade-off discussed in Section F.2.

These results establish a coupling between aggregated experience quality and bootstrapping behaviour: better trajectories yield better state knowledge, and the agent's ability to extract predictive insights tracks this quality closely. **When the agent must actively decide what futures to simulate, it is continually forced to balance exploring uncertain branches against exploiting known strong lines—a trade-off it learns to navigate autonomously.**

#### 4.2.2 EXPERIMENTS WITH SELF-GENERATED PARALLEL SIMULATION

We now let the agent decide what to simulate by itself, jointly optimizing experience generation and the bootstrapping loop end-to-end. A complete reasoning trace for SKL-RL under self-generated simulation is provided in Appendix K.4. We use SKL-RL with parallel search at a budget of $N = 3, H = 2$, denoted as SKL-RL$_{N3H2}$. GRPO is applied for training on the full SKL-RL track under loss $\mathcal{L}_{\text{GRPO}}(\mathcal{D}_{\text{SKL-RL}})$.

Figure 4 compares SKL-RL against existing training baselines under matched training budgets. SKL-RL$_{N3H2}$ trained end-to-end achieves the strongest performance throughout training and the highest test-time few-shot accuracy. In contrast, naive GRPO and Reflect-GRPO (Bensal et al., 2025) plateau on the training set and show noticeable performance degradation at test time. Critic-GRPO (Zhang et al., 2025b) suffers from training instability, leading to collapse. Although R$^3$L achieves the best performance among the remaining baselines, it still exhibits a slight drop on the held-out test set.

Also observed in Figure 4, both the diversity of simulated moves and the response length in the bootstrapping loop undergo clear fluctuations. Here we provide one possible explanation. The sharp

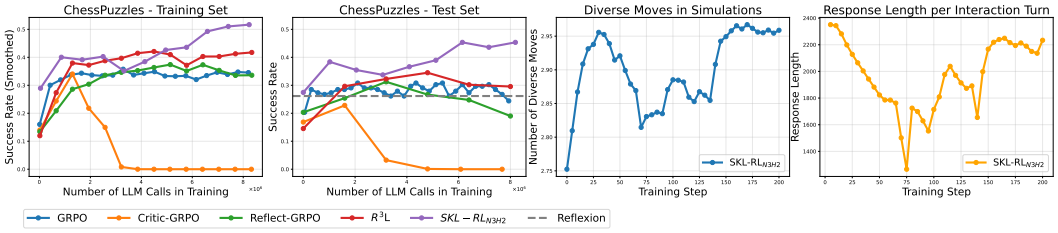



Figure 4: SKL-RL training with self-generated parallel simulation, evaluated at matched budgets.

drop of both metrics at around training step 75 suggests an early synchronized adjustment between diversity and exploitation, while the gradual increase after step 75 for both move diversity and response length indicates that the agent automatically allocates a larger reasoning budget to support broader exploration, enabling more effective extraction of information from simulated experience. This empirical result directly echoes our previous assumption about the emergent self-adaptive behaviour during end-to-end training.

When comparing trained reasoning traces (Appendix K.5), SKL-RL seems to produce more outcome-oriented understanding than R$^3$L. Empirically speaking, the agent moves away from chess-specific clichés and instead grounds decisions in implicit evaluative signals such as material balance and concrete tactics, while trajectory-level reflection remains coarser and path-dependent.

#### 4.2.3 Experiments with stateful knowledge distillation

We now examine how state-knowledge distillation guides policy learning, thereby closing the learning loop in SKL-RL. The self-distillation loss $\mathcal{L}_{\text{Distill}}(\{z_s, \hat{z}_s\}_{s\in\mathcal{S}})$ internalizes post-hoc stateful knowledge into the prior state understanding, enabling the agent to anticipate predictive knowledge for future training cycles or to directly produce sophisticated reasoning plans at test time. To isolate its effect while reducing computational cost, we shorten the ChessPuzzles episode length to 5 and ablate the loss coefficient $\lambda$ and the update interval $\delta_n$, where distillation is applied every $n$ training steps (Figure 5).

We first observe that removing $\mathcal{L}_{\text{Distill}}$ leads to noticeable performance degradation, consistent with the trend in Figure 4, and that this instability worsens as episode length decreases. In contrast, increasing $\lambda$ and applying distillation updates more frequently markedly stabilizes training. Similar to recent works, we also find that the reverse KL, a multi-turn variant of *On-Policy Self-Distillation* (Zhao et al., 2026; Hübotter et al., 2026), provides greater stability than the forward KL setting.

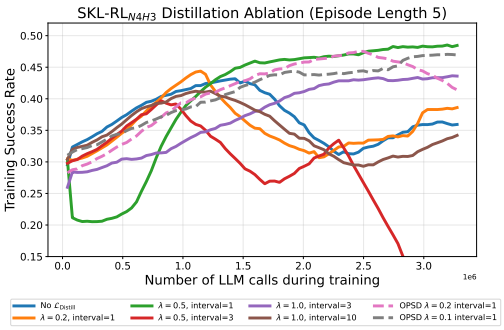



Figure 5: Ablation study on SKL-RL with integrated training loss.

These results suggest that $\mathcal{L}_{\text{Distill}}$ can help stabilize the joint optimization of this compositional reasoning process. As shown by the reasoning traces in Appendix K.6, the SKL-RL agent **updates stateful predictive knowledge by building upon previously distilled information**. Specifically, it treats $\hat{z}$ from the prior learning epoch as a condensed representation, reusing it to empower the agent to explore and exploit more effectively via continuously accumulated stateful knowledge.

**The SKL-RL framework exhibits several compelling properties**: it significantly reduces reliance on human intervention and complex pipeline engineering, enables self-adaptive behaviours to emerge purely from end-to-end training, and demonstrates a distinct capacity to build new knowledge incrementally on top of prior iterations. Despite these strengths, in its minimal implementation, SKL-RL faces severe **practical limitations** in computational efficiency relative to current state-of-the-art self-evolving agents. This is primarily due to the substantial computational cost of state-wise simulation, which leads to overly long training samples and, in our independent-branch implementation, approximately $O(TNH)$ simulated transitions for an episode of length $T$. Nevertheless, this exploration

validates our initial vision for next-generation autonomous agents: systems capable of naturally extracting predictive information from environmental dynamics, absorbing it into a persistent internal 'mental map', and dynamically balancing exploration and exploitation in accordance with their own evolving capabilities.

## 5 Related works

**Trajectory-level Self-Reflection.** A broad line of work improves LLM agents through reasoning and reflection over interaction trajectories. ReAct (Yao et al., 2022b) introduced the thought–action–observation paradigm, while Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use generated feedback to revise subsequent behavior. Recent post-training methods further learn from critiques, retries, or improved responses (Wang et al., 2025; Sun et al., 2023; Wan et al., 2025; Zhang et al., 2025b; Bensal et al., 2025; Shi et al., 2026b). These methods may reuse one or multiple trajectories; the distinction from SKL is therefore not reuse frequency, but the update target and organization of evidence. Trajectory-level methods typically refine a global critique, plan, or response, whereas SKL groups continuations by their originating state and iteratively updates state-indexed predictive knowledge, providing finer-grained guidance at intermediate decisions (Wang et al., 2026b).

**State-grounded Memory and Value Representations.** State-dependent representations are well established in structured memory systems (Wang et al., 2026a; Zhou et al., 2025; Wang, 2025; Zhou et al., 2026) and reinforcement-learning-inspired agents using scalar state or state–action values (Yan et al., 2024; Zhang et al., 2023; Zhai et al., 2025; Yan et al., 2025; Zhang et al., 2026; Wang et al., 2026b). We therefore do not claim that operating at the state level is itself novel. SKL instead maintains a self-generated natural-language assessment $z_s$ that predicts downstream outcomes and directly conditions action selection. Unlike a scalar value, $z_s$ can retain the semantic reasons behind its assessment. The closest direction is the *language value function* of Feng et al. (2024); SKL extends this idea through state-matched aggregation, **Knowledge Bootstrapping** from successor assessments, and internalization through SKL-SD and SKL-RL.

**World Models, Search, and Planning.** World-model-based agents reason over predicted transitions (Yu et al., 2025; Guo et al., 2025b; Zhang et al., 2025a; Chu et al., 2026), while search-based methods explicitly explore alternative continuations (Wu et al., 2026; Liu et al., 2026; Wang et al., 2026b). SKL similarly uses future outcomes to assess the current state, but does not learn a full transition model or optimize explicit one-step prediction error. Instead, it can be viewed through the lens of value equivalence (Grimm et al., 2020) under a semantic Bellman-consistency assumption. In SKL-RL, parallel simulations provide state-anchored evidence that is synthesized into reusable language-valued knowledge, rather than serving only as immediate search branches or scalar backups.

## 6 Conclusions and limitations

In this paper, we have challenged the prevailing reliance on solely trajectory-level reflection in LLM agent training. To address this, we introduced **Stateful Knowledge Learning (SKL)**, a framework that centers on maintaining explicit, declarative predictive assessments anchored to specific states. Our motivating examples confirmed that stateful knowledge provides finer granularity and generalization, and also enables knowledge bootstrapping that scales SKL to increasingly complex tasks. Experiments validate how SKL training variants **SKL-SD** and **SKL-RL** can potentially lead to performance gains and move toward a more autonomous, self-evolving agent learning paradigm.

As LLM agents transition further into the "Era of Experience", moving beyond static datasets toward continuous environmental interaction, the ability to maintain a stateful "mental map" of predictive foresight will be critical. While our current implementation introduces a trade-off in training compute, the resulting gains in policy robustness and strategic depth suggest that stateful learning provides a structured framework for organizing and internalizing self-reflection. Future work will explore the practical implementation to resolve the computational limitation raised in the paper and the potential for transferring stateful predictive knowledge across heterogeneous task domains.

### AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for [ help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs]. We have not used generative AI tools for [propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, interpret results], and [Generate synthetic data sets, assist with translation, clean and reformat dataset, ] are not applicable to this work. Additionally, we used generative AI tools for [create or edit software code, edit a research paper to improve readability,]. We have reviewed all AI-assisted work. [LLM-generated code was verified and tested for correctness by 2 authors]. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

See the ICLR 2027 AI Policy for Authors for more details. This statement should not be more than 1 page.

## REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The twelfth international conference on learning representations*, 2024.

Shelly Bensal, Umar Jamil, Christopher Bryant, Melisa Russak, Kiran Kamble, Dmytro Mozolevskyi, Muayad Ali, and Waseem AlShikh. Reflect, retry, reward: Self-improving llms via reinforcement learning. *arXiv preprint arXiv:2505.24726*, 2025.

Ching-An Cheng, Andrey Kolobov, Dipendra Misra, Allen Nie, and Adith Swaminathan. Llf-bench: Benchmark for interactive learning from language feedback. *arXiv preprint arXiv:2312.06853*, 2023.

Meng Chu, Xuan Billy Zhang, Kevin Qinghong Lin, Lingdong Kong, Jize Zhang, Teng Tu, Weijian Ma, Ziqi Huang, Senqiao Yang, Wei Huang, et al. Agentic world modeling: Foundations, capabilities, laws, and beyond. *arXiv preprint arXiv:2604.22748*, 2026.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

Xidong Feng, Bo Liu, Yan Song, Haotian Fu, Ziyu Wan, Girish A Koushik, Zhiyuan Hu, Mengyue Yang, Ying Wen, and Jun Wang. Natural language reinforcement learning. *arXiv preprint arXiv:2411.14251*, 2024.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *Advances in neural information processing systems*, 33: 5541–5552, 2020.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Shangmin Guo, Omar Darwiche Domingues, Raphaël Avalos, Aaron Courville, and Florian Strub. Sample, predict, then proceed: Self-verification sampling for tool use of llms. *arXiv preprint arXiv:2506.02918*, 2025b.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Dongyoon Hwang, Hojoon Lee, Jaegul Choo, Dongmin Park, and Jongho Park. Can large language models develop strategic reasoning? post-training insights from learning chess. *arXiv preprint arXiv:2507.00726*, 2025.

Michael Littman and Richard S Sutton. Predictive representations of state. *Advances in neural information processing systems*, 14, 2001.

Anjie Liu, Ziqin Gong, Yan Song, Yuxiang Chen, Xiaolong Liu, Hengtong Lu, Kaike Zhang, and Chen Wei. Active reasoning vision-language models via sequential experimental design. 2026. URL https://api.semanticscholar.org/CorpusID:287948114.

Jincheng Liu, Sijun He, Jingjing Wu, Xiangsen Wang, Yang Chen, Zhaoqi Kuang, Siqi Bao, and Yuan Yao. Chessarena: A chess testbed for evaluating strategic reasoning capabilities of large language models. *arXiv preprint arXiv:2509.24239*, 2025.

Weiyu Ma, Yongcheng Zeng, Yan Song, Xinyu Cui, Jian Zhao, Xuhui Liu, and Mohamed Elhoseiny. Freshness-aware prioritized experience replay for llm/vlm reinforcement learning. *arXiv preprint arXiv:2604.16918*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Xuchen Pan, Yanxi Chen, Yushuo Chen, Yuchang Sun, Daoyuan Chen, Wenhao Zhang, Yuexiang Xie, Yilun Huang, Yilei Zhang, Dawei Gao, et al. Trinity-rft: A general-purpose and unified framework for reinforcement fine-tuning of large language models. *arXiv preprint arXiv:2505.17826*, 2025.

Tord Romstad, Marco Costalba, Joona Kiiski, Gary Linscott, Yu Nasu, Motohiro Isozaki, Hisayori Noda, and The Stockfish Development Team. Stockfish, 2024. URL https://stockfishchess.org/.

Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li K Wenliang, Elliot Catt, John Reid, Cannada A Lewis, Joel Veness, and Tim Genewein. Amortized planning with large-scale transformers: A case study on chess. *Advances in Neural Information Processing Systems*, 37:65765–65790, 2024.

Taiwei Shi, Sihao Chen, Bowen Jiang, Linxin Song, Longqi Yang, and Jieyu Zhao. Experiential reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.13949.

Weijie Shi, Yanxi Chen, Zexi Li, Xuchen Pan, Yuchang Sun, Jiajie Xu, Xiaofang Zhou, and Yaliang Li. R $^3$ l: Reflect-then-retry reinforcement learning with language-guided exploration, pivotal credit, and positive amplification. *arXiv preprint arXiv:2601.03715*, 2026b.

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

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768, 2020. URL https://api.semanticscholar.org/CorpusID:222208810.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1, 2025.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245, 2023.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th international conference on autonomous agents and multiagent systems-volume 2*, pp. 761–768, 2011.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, et al. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *arXiv preprint arXiv:2503.09501*, 2025.

Jun Wang. Memento-ii: Learning by stateful reflective memory. *arXiv preprint arXiv:2512.22716*, 2025.

Juyuan Wang, Rongchen Zhao, Wei Wei, Yufeng Wang, Mo Yu, Jie Zhou, Jin Xu, and Liyan Xu. Comorag: A cognitive-inspired memory-organized rag for stateful long narrative reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33557–33565, 2026a.

Zehong Wang, Fang Wu, Hongru Wang, Xiangru Tang, Bolian Li, Zhenfei Yin, Yijun Ma, Yiyang Li, Weixiang Sun, Xiusi Chen, et al. Why reasoning fails to plan: A planning-centric analysis of long-horizon decision making in llm agents. *arXiv preprint arXiv:2601.22311*, 2026b.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Fang Wu, Weihao Xuan, Heli Qi, Aaron Tu, Ximing Lu, Li Li, and Yejin Choi. Deepsearch: Overcome the bottleneck of reinforcement learning with verifiable rewards via tree-based search. In *International Conference on Learning Representations*, volume 2026, pp. 95559–95584, 2026.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Kristian Kersting, Jeff Z Pan, Hinrich Schütze, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

Xue Yan, Yan Song, Xidong Feng, Mengyue Yang, Haifeng Zhang, Haitham Bou Ammar, and Jun Wang. Efficient reinforcement learning with large language model priors. *arXiv preprint arXiv:2410.07927*, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chaorui Yao, Yanxi Chen, Yuchang Sun, Yushuo Chen, Wenhao Zhang, Xuchen Pan, Yaliang Li, and Bolin Ding. Group-relative reinforce is secretly an off-policy algorithm: Demystifying some myths about grpo and its friends. *arXiv preprint arXiv:2509.24203*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *ArXiv*, abs/2207.01206, 2022a. URL `https://api.semanticscholar.org/CorpusID:250264533`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022b.

Xiao Yu, Baolin Peng, Ruize Xu, Michel Galley, Hao Cheng, Suman Nath, Jianfeng Gao, and Zhou Yu. Dyna-think: Synergizing reasoning, acting, and world model simulation in ai agents. *arXiv preprint arXiv:2506.00320*, 2025.

Linda Zagzebski. What is knowledge? *The Blackwell guide to epistemology*, pp. 92–116, 2017.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36:78227–78239, 2023.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, et al. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*, 2025a.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025b.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, et al. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

# A ALGORITHM PSEUDOCODE

**Algorithm 1 State-Knowledge-Learning Self-Distillation** (SKL-SD)

**Require:** Policy $\pi_\theta$, Environment $E$, Outcome verifier $R$, Bootstrap budget $(N, H)$
**Ensure:** $\pi_\theta$

Initialize global experience buffer $\mathcal{B} \leftarrow \emptyset$
**for** Epoch $e = 1$ to $K$ **do**
  Initialize dataset $\mathcal{D}_{\text{base}}, \mathcal{D}_{\text{distill}}, \mathcal{D}_{\text{reflect}} \leftarrow \emptyset, \emptyset, \emptyset$
  *# Base Stateful Rollout*
  Sample base stateful trajectories in batches: $\boldsymbol{\tau}^e = \{\tau_1, \tau_2, ..., \tau_N\} \sim \pi_{\theta_{\text{old}}}$
  Add transition to global experience buffer $\mathcal{B} \leftarrow \tau^e$
  **for** each base trajectory $\tau_i$ in $\boldsymbol{\tau}^e$ **do**
    **if** $R(\tau_i) > 0$ **then**
      $\mathcal{D}_{\text{base}} \leftarrow \mathcal{D}_{\text{base}} \cup \tau_i$
    **else**
      *# Bootstrapping on failed trajectories*
      **for** timestep $t = T - 1$ to $0$ **do**
        Sample aggregated trajectories $\{\tau_n^H(s_t^{(i)})\}_{n=1}^N \sim \mathcal{B}$ originated at state $s_t^{(i)} \in \tau_i$
        Generate updated state knowledge $\hat{z}^e_{s_t^{(i)}} \sim \pi_{\theta_{\text{old}}}\big(\cdot \,\big|\, s_t^{(i)}, \{\tau_n^H(s_t^{(i)})\}_{n=1}^N, z^{e-1}_{s_t^{(i)}}, z^*_{s_{t+H}^{(i)}}\big)$
      **end for**
      *# Verify updated state knowledge via Rejection Sampling*
      Sampling retried stateful trajectory $\tau_i^{\text{retry}}$ with pre-defined state knowledge $\{\hat{z}^e_{s_t^{(i)}}\}_{t=0}^{T-1}$
      **if** $R(\tau_i^{\text{retry}}) > 0$ **then**
        $\mathcal{D}_{\text{distill}} \leftarrow \mathcal{D}_{\text{distill}} \cup \tau_i^{\text{retry}}, \mathcal{D}_{\text{reflect}} \leftarrow \mathcal{D}_{\text{reflect}} \cup \big(\big[s_t^{(i)}, \{\boldsymbol{\tau}_n^H(s_t^{(i)})\}_{n=1}^N, z^{e-1}_{s_t^{(i)}}, z^*_{s_{t+H}^{(i)}}\big], \hat{z}^e_{s_t^{(i)}}\big)$
      **end if**
    **end if**
    *# Policy Update with SKL-SD Objective Function*
    $\mathcal{L}_{\text{SKL-SD}}(\theta) = \mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{base}}) + \lambda_1 \mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{distill}}) + \lambda_2 \mathcal{L}_{\text{SFT}}(\mathcal{D}_{\text{reflect}})$
    $\theta_{\text{old}} \leftarrow \arg\min_\theta(\mathcal{L}_{\text{SKL-SD}}(\theta))$
  **end for**
**end for**

**Algorithm 2 State-Knowledge-Learning Reinforcement Learning** (SKL-RL)

**Require:** Policy $\pi_\theta$, Environment $E$, Simulator $E^{\text{sim}}$, Outcome verifier $R$, Bootstrap budget $(N, H)$
**Ensure:** $\pi_\theta$

**for** Epoch $e = 1$ to $K$ **do**
  Initialize dataset $\mathcal{D}_{\text{SKL-RL}}, \mathcal{D}_{\text{distill}} \leftarrow \emptyset, \emptyset$
  *# Aggregated Stateful Rollout via Simulation*
  **for** $t = 0$ to $T - 1$ **do**
    Simulate $N$ parallel stateful trajectories $\boldsymbol{\tau}_N^H(s_t) = \{\tau_1^H(s_t), ..., \tau_N^H(s_t)\}$ with horizon $H$
    Generate updated state knowledge $\hat{z}_{s_t} \sim \pi_{\theta_{\text{old}}}(\cdot|s_t, z_{s_t}, \boldsymbol{\tau}_N^H(s_t))$
    Generate exploiting action $a_t \sim \pi_{\theta_{\text{old}}}(\cdot|s_t, \hat{z}_{s_t})$
    Transition $s_{t+1} \sim E(s_t, a_t)$, $s_t \leftarrow s_{t+1}$
  **end for**
  Collect trajectories $\boldsymbol{\tau}^e = [s_0, (z_{s_0}, \boldsymbol{\tau}_N^H(s_0), \hat{z}_{s_0}, a_0), s_1, (z_{s_1}, \boldsymbol{\tau}_N^H(s_1), \hat{z}_{s_1}, a_1), ..., s_T]$
  $\mathcal{D}_{\text{SKL-RL}} \leftarrow \boldsymbol{\tau}^e$, $\mathcal{D}_{\text{distill}} \leftarrow \{(z_{s_t}, \hat{z}_{s_t})\}_{t=0}^{T-1}$
  *# Policy Update with SKL-RL Objective Function*
  $\mathcal{L}_{\text{SKL-RL}}(\theta) = \mathcal{L}_{\text{GRPO}}(\mathcal{D}_{\text{SKL-RL}}) + \lambda \mathcal{L}_{\text{Distill}}(\mathcal{D}_{\text{distill}})\,, \quad \theta_{\text{old}} \leftarrow \arg\min_\theta(\mathcal{L}_{\text{SKL-RL}}(\theta))$
**end for**

[*] $z_{s_{t+H}^{(i)}}$ can be retrieved from the buffer or the previous time-step in the same bootstrapping loop.

## B NOTATION

| Symbol | Description |
|---|---|
| $E$ | Environment |
| $E^{\text{sim}}$ | A resettable environmental simulator |
| $\mathcal{S}$ | Environmental state space |
| $s_t$ | Environmental state at time $t$ |
| $\pi_\theta$ | Policy function with parameter $\theta$ |
| $\mathcal{A}$ | Action space |
| $a_t$ | Executable action at time $t$ |
| $z_{s_t}$ | Generated state knowledge at state $s_t$ |
| $\{z_s\}_{s\in\mathcal{S}}$ | State knowledge table |
| $z(\cdot)$ | Generation function for state knowledge |
| $\hat{z}_{s_t}$ | Updated state knowledge at state $s_t$ |
| $(H, N)$ | Bootstrap budget for state aggregation; $H$ is the maximum horizon, $N$ is the number of rollout branches |
| $\tau(s_0)$ | A complete rollout trajectory starting at initial state $s_0$ |
| $\{\tau_n^H(s_t)\}_{n=1}^N$ | $N$ number of sampled partial rollout trajectories originated at $s_t$ with budget $(H, N)$, used in SKL-SD offline bootstrapping stage |
| $\boldsymbol{\tau}_N^H(s_t)$ | A set online model-based rollout trajectories originated at $s_t$ with budget $(H, N)$, used in SKL-RL online interaction stage |
| $\mathcal{B}$ | Experience buffer |
| $R, R(\cdot)$ | Outcome reward (function) |

Table 2: Notation Description

## C FORMAL DEFINITION

We use "stateful" in a similar sense as classical reinforcement learning value functions: a value function $V(s_t)$ is stateful as it evaluates the expected future return starting from a particular environmental state $s_t \in \mathcal{S}$, as opposed to an entire complete trajectory. Same for our stateful predictive knowledge: it is indexed by the current environmental state and summarises what that state implies for future outcomes.

We define Stateful Predictive Knowledge as a policy-dependent, language-valued mapping:

$$z^\pi : \mathcal{S} \to \mathcal{L}_{\text{lang}}, \quad s_t \mapsto z^\pi_{s_t}.$$

$z^\pi_{s_t}$ has three defining properties:

*State-indexed*: it is stored, updated, and retrieved using the current environmental state $s_t$ as its anchor. *Predictive*: it describes likely downstream outcomes, risks, opportunities, or bottlenecks under possible continuations from $s_t$, rather than merely restating the current observation. *Decision-relavant*: it is used to select the next action, so that $a_t \sim \pi_\theta(a_t \mid s_t, z^\pi_{s_t})$.

This is deliberately value-like, but it is not just a scalar value written in words. A scalar $V^\pi(s_t) \in \mathbb{R}$ compresses future utility into one scalar number, whereas $z^\pi_{s_t}$ can retain the semantic reasoning behind that assessment, such as which action creates a tactical risk, which constraint is still unsatisfied, or which successor state is difficult to recover from. It is also not free-form reflection over an entire failed attempt: the update target and the evidence are both anchored at $s_t$.

Because language cannot be averaged or discounted directly, we use an LLM-parameterized Semantic Aggregation Operator $\mathbb{G}_\theta(\cdot)$. It maps sampled evidence to a unified declarative assessment through LLM inference, bootstrapping the stateful understanding from outcome to root state. We named it "Knowledge Bootstrapping". The framework supports two iterative update styles:

**Monte Carlo-style update** . Let $\{\boldsymbol{\tau}^n_{1:\mathrm{T}}(s_t)\}_{n=1}^N$ be a set of $T$ forward trajectories sampled from $s_t$ until termination or a heuristic stopping point. - Then:

$$z^{\mathrm{MC},\pi}_{s_t} = \mathbb{G}_\theta\left(s_t, z^{\mathrm{old}}_{s_t}, \{\boldsymbol{\tau}^n_{1:T}(s_t)\}_{n=1}^N\right)$$

is the semantic assessment of $s_t$ based on previous assessment and $N$ subsequent outcomes.

**Temporal-difference-style update** . Let the simulations instead stop after a local horizon $H$:

$$\{\boldsymbol{\tau}^n_{1:\mathrm{H}}(s_t)\}_{n=1}^N = (s_t, a^{(n)}_t, \ldots, s^{(n)}_{t+H})_{n=1}^N.$$

The current state assessment is updated using both this local evidence and the previously computed knowledge at the successor states:

$$z^{\mathrm{TD},\pi}_{s_t} = \mathbb{G}_\theta\left(s_t, z^{\mathrm{old}}_{s_t}, \{\boldsymbol{\tau}^n_{1:H}(s_t)\}_{n=1}^N, \quad \bigoplus_{n=1}^{N} z^{\mathrm{old}}_{s^{(n)}_{t+H}}\right)$$

where $\bigoplus$ denotes concatenation followed by semantic synthesis, not numerical summation.

This definition also reveals how statefulness is enforced by constraining the context input. First, experiences from different trajectories are grouped by the state from which they continue. Second, the semantic aggregation operator $\mathbb{G}_\theta(\cdot)$ receives only the current state and the local continuations; Third, the resulting knowledge is reused to give a valid action for current state transition. This is a Markov-style design constraint in language space: any past information needed for decision-making should already be encoded in the state representation $s_t$. In partially observable settings, $s_t$ can be interpreted as a belief state or task-state summary.

## D Extra experiments on Slippery FrozenLake

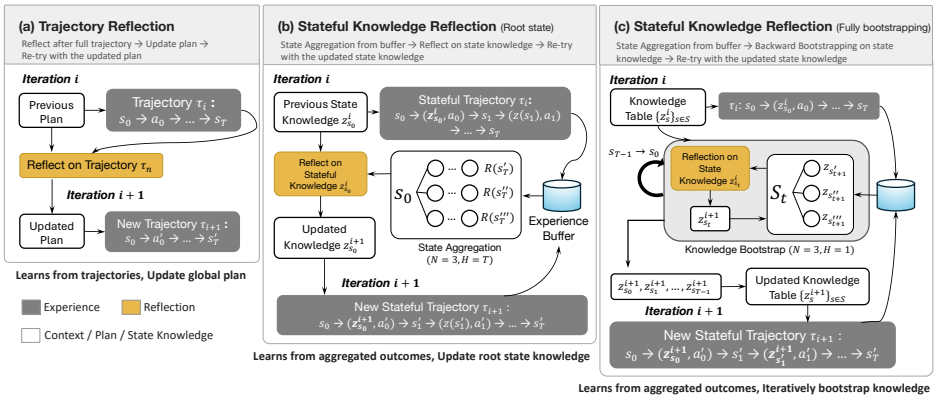



Figure 6: **Trajectory-level reflection and Stateful Knowledge Reflection** (root-state and fully bootstrapping variants). The distinction is *which representation is indexed and updated*. Trajectory reflection uses one or more completed trajectories to update a trajectory-level plan or critique. Stateful Knowledge Reflection groups continuations by their originating state $s_t$ and updates the corresponding predictive knowledge $z_{s_t}$; the fully bootstrapping variant further propagates successor-state assessments backward.

In Section 2, we compared trajectory-level reflection with Stateful Knowledge Reflection. Here, we present a complementary experiment where we allow the trajectory reflection mechanism to process multiple past trajectories simultaneously, aligning it more closely with the state aggregation used in the stateful agent setup. While this broader context exposes the agent to richer information regarding the true environmental dynamics, it also requires the agent to extract valuable insights from a much noisier experience buffer. As shown in Figure 7, reading multiple trajectories does improve the agent's understanding of the environment, particularly in the asymmetric scenario ($P = (0, 2/3)$), but the agent still fails to generalize effectively to the testing set. Table 3 provides an example of a reflection trace on the same map used in Figure 1. It demonstrates that while trajectory reflection with multiple reads successfully discovers the asymmetric dynamics, it only yields a coarse approximation.

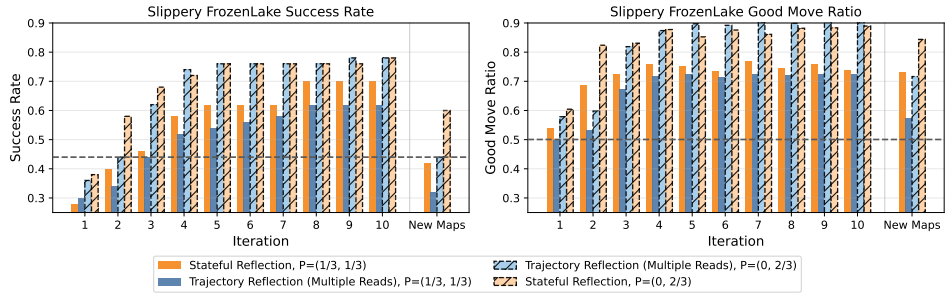



Figure 7: **Extra experiments on Slippery FrozenLake:** Comparison of trajectory-level with multiple reads and stateful knowledge reflection. Unlike the setup in Figure 1, the trajectory reflection mechanism here processes multiple past trajectories simultaneously when updating its plans. While this enables better performance compared to reflecting strictly on the most recent trajectory, the agent still struggles to transfer these extracted insights to testing scenarios.

Table 3: Trajectory Reflection with Multiple Reads at final iteration. ($P = (0, 2/3)$)

**Map**:
___O_
OP___
O_OO_
_O__G

**Final reflection with multiple reads**:
Upon reviewing my previous attempt on the FrozenLake map, I recognize that I underestimated the true slip probability and failed to account for environmental dynamics properly. I assumed that each action had an equal chance of slipping in any direction (as per the typical 1/3 slip model), but the observed outcomes suggest a higher likelihood of unintended forward movement when attempting lateral actions — indicating a directional bias in the slip dynamics. **(Successfully discover the abnormal dynamics)** Specifically, when I attempted to move left (West) from a critical tile adjacent to a hole, I ended up moving forward (North) twice in a row, falling into the hole. This occurred despite no intended "North" action, suggesting that the slip distribution is skewed — possibly 50% intended direction, 30% forward (relative to previous direction), 20% other directions, rather than uniform. (**try to infer underneath dynamics**, **but the guess is poor**)
My mistake was relying on a greedy path without building in recovery steps or safe detours. I attempted a high-risk route with minimal buffer zones and did not use repeated safe-state verification via consistent backtracking or environment probing.

**New Plan**: Six-Point Strategy
- *Verify Slip Dynamics Proactively ...*
- *Prefer Diagonal Avoidance Over Direct Lateral Moves Near Holes ...*
- *Use High-Confidence Anchors ...*
- *Adopt a Right-Hand Wall-Following Heuristic with Recovery ...*
- *Commit to the Path Only After Simulated Recovery ...*
- *Exploit Deterministic Recovery Zones ...*

**Key correction**: Never move laterally adjacent to a hole without a one-tile buffer and a pre-planned escape action.
.....

# E General Summary of the Essence of Stateful Predictive Knowledge

Fundamentally, the essence of SKL is to extract actionable insights from past experience to guide future decision-making. To achieve this, the framework must balance two competing properties of the extracted knowledge: **Representation** (the breadth of empirical information collected) and **Regularization** (whether the knowledge remains temporally consistent).

**Controlling Representation: State Aggregation  Parallel Simulation.** These two components serve the exact same fundamental purpose: gathering multiple trajectory rollouts that branch from the same root state to provide diverse empirical evidence. State Aggregation is the process of grouping these state-anchored experiences together. Parallel Simulation is simply the online, active-search equivalent of state aggregation (utilized specifically in our SKL-RL algorithm), where the agent deliberately rolls out multiple parallel futures on the fly. More state aggregation/simulation means more rollout information, which increases the representation power. However, as demonstrated later in our ablation studies, an "overdose" of rollout information can lead to redundancy and distract the agent. Thus, their contribution is to provide the raw empirical signal, which must be carefully calibrated.

**Enforcing Regularization: Knowledge Bootstrapping.** Bootstrapping enforces temporal consistency across the states via the Semantic Bellman Equation. By tying the knowledge of state to the knowledge of , bootstrapping regularizes the representations to ensure they satisfy the Markov property in language space. As also shown in our additional ablation studies, scaling the bootstrapping budget also exhibits a trade-off, meaning regularization must be dynamically balanced against the raw representation power.

**The Synergy: How SKL-SD and SKL-RL Balance the Trade-off.** The exact contribution of these components depends on how the agent is allowed to balance them: In our later additional ablation for SKL-SD (Appendix M): We intend to isolate the effect of regularization. We will fix the representation power (by locking the state aggregation budgets, and ) and explore the scaling law of regularization by ablating the fraction of states that undergo bootstrapping and report token usage. In the SKL-RL experiments, the agent dynamically determines its own data generation pipeline. By utilizing self-generated parallel simulations, the agent jointly optimizes representation power and bootstrapping regularization end-to-end. This joint optimization is exactly what drives the self-adaptive behavior detailed in Section 4.2.2.

**Internalization: Distillation** Distillation operates entirely downstream of the aforementioned synergy. Its sole contribution is internalization, that is, baking the extracted, regularized stateful knowledge into the LLM's parametric weights so the agent can learn incrementally without relying on expanding inference-time context windows.

# F Details of SKL-SD and SKL-RL

## F.1 State-Knowledge-Learning Self-Distillation (SKL-SD)

SKL-SD extracts stateful knowledge through model-free rollout and bootstrapping, internalizing state-aware behaviour via quality-filtered self-distillation. Drawing inspiration from recent self-reflection paradigms (Zhang et al., 2025b; Shi et al., 2026b), we construct a multi-faceted dataset $\mathcal{D}_{\text{SKL-SD}} = [\mathcal{D}_{\text{base}}, \mathcal{D}_{\text{distill}}, \mathcal{D}_{\text{reflect}}]$ generated through four iterative steps:

**I. Model-free Stateful Rollout**: The agent performs knowledge-grounded reasoning to generate trajectories $\tau = (s_0, (z_{s_0}, a_0), s_1, (z_{s_1}, a_1), ..., s_T, R)$. Successful trajectories are added to $\mathcal{D}_{\text{base}}$, while all trajectories are stored in an experience buffer $\mathcal{B}$ for subsequent state aggregation.

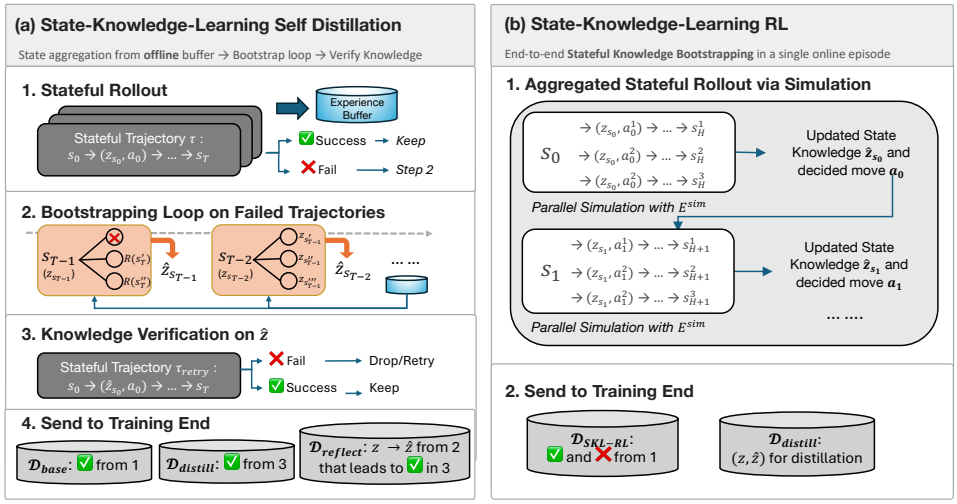



Figure 8: **Illustration of two training variants: SKL-SD and SKL-RL**. SKL-SD performs bootstrapping on initial failures to collect successful secondary rollouts. SKL-RL performs parallel simulations for each state in an episode to update state knowledge and identify an exploitative first move, helping the agent transition to the next state in the environment.

**II. Bootstrapping Loop with a Replay Buffer**: On failed trajectories, the agent initiates a backward bootstrapping loop from $s_{T-1}$ back to $s_0$. It sequentially updates the state knowledge by aggregating outcomes from the experience buffer $\{\tau_n^H(s_t)\}_{n=1}^N \sim \mathcal{B}$ as well as the successor assessments to construct informative context $\mathcal{C}_t$:

$$\mathcal{C}_t = [s_t, \{\tau_n^H(s_t)\}_{n=1}^N, z_{s_t}, z_{s_{t+H}}] \tag{3}$$

where $s_t$ refers to the target state being evaluated, $\{\tau_n^H(s_t)\}_{n=1}^N$ refers to a set of aggregated partial rollouts sampled from the experience buffer, all originating at $s_t$, at bootstrapping budget $(H, N)$. $z_{s_t}$ refers to the current knowledge (prior belief) associated with the target state, and $z_{s_{t+H}}$ represents the successor stateful knowledge, providing the bootstrapping signal from $H$ steps ahead [2]. The agent then generates refined knowledge $\hat{z}_{s_t} \sim \pi_\theta(\cdot|\mathcal{C}_t)$, resulting in a sequence $\hat{\boldsymbol{z}} = (\hat{z}_{s_0}, ..., \hat{z}_{s_{T-1}})$.

**III. Knowledge Verification**: We verify $\hat{\boldsymbol{z}}$ by performing a second rollout using the refined knowledge as a pre-filled context. Successful second rollouts will be added to $\mathcal{D}_{\text{distill}}$ and the corresponding state-wise update $\boldsymbol{z} \to \hat{\boldsymbol{z}}$ correcting the first failure is recorded in $\mathcal{D}_{\text{reflect}}$.

**IV. Internalize Knowledge to Guide Policy**: The agent is fine-tuned by maximizing the likelihood of these traces using policy-gradient loss (e.g., GRPO (Guo et al., 2025a)) on the full trajectory. We also incorporate an auxiliary SFT loss to enhance state-wise bootstrapping capabilities:

$$\mathcal{L}_{\text{SKL-SD}} = \mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{base}}) + \lambda_1 \cdot \mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{distill}}) + \lambda_2 \cdot \mathcal{L}_{\text{SFT}}(\mathcal{D}_{\text{reflect}}) \tag{4}$$

Intuitively, SKL-SD can be viewed as a "stateful" version of recent self-reflection training methods with data filtering (Appendix G), aiming to maximize the token probability of the first and the second stateful rollout if they hit the target with or without knowledge bootstrapping, and reinforce the bootstrapping behaviour itself if it successfully helps the agent correct its mistake. We name it "Self-Distillation" (or to be more precise "Self-Imitation") as it essentially performs supervised learning over heuristically quality-filtered experience.

### F.2 State-Knowledge-Learning Reinforcement Learning (SKL-RL)

SKL-RL serves as the online counterpart to SKL-SD, interacting with the simulator $E^{\text{sim}}$ and extracting stateful knowledge entirely within an online rollout episode, linking the quality of the

[2] $z_{s_{t+H}}$ can be retrieved from experience $\boldsymbol{\tau}$ or from the updated knowledge $\hat{z}_{s_{t+H}}$ within the same bootstrapping loop

bootstrapping loop directly to the task outcome. In this variant, the agent is assumed to have access to a state-settable environmental simulator, performing **localized parallel simulations** (Appendix H) at each state $s_t$ to collect *real-time* aggregated experiences before synthesizing updated knowledge and committing to an action.

**I. & II. Online Model-based Rollout and Bootstrapping Loop**: At each environmental time-step $t$, the agent first interacts with simulator $E^{\text{sim}}$ and generates multiple look-ahead rollouts $\boldsymbol{\tau}_N^H(s_t) = \{s_t, (z_{s_t}, a_t^{(n)}), s_{t+1}^{(n)}, ..., s_{t+H}^{(n)}\}_{n=1}^N$. These trajectories, operating under a bootstrapping budget of $(H, N)$, are all rooted at the current state $s_t$ and can be sampled via heuristic policies or from the agent's own knowledge-based understanding. These localized parallel simulations provide the necessary context for a real-time bootstrapping step, synthesizing refined state knowledge $\hat{z}_{s_t}$ to ground the final action selection $a_t$ for the real environment:

$$\hat{z}_{s_t} \sim \pi_\theta(\cdot|\boldsymbol{\tau}_N^H(s_t), s_t, z_{s_t}) \,,\ a_t \sim \pi_\theta(\cdot|\hat{z}_{s_t}, s_t) \,,\ \ t \to t+1 \tag{5}$$

Appendix H and Figure 10 provide a concrete example for illustration. The complete simulation-augmented trajectory can be denoted as:

$$\cdots \to s_t \to \boldsymbol{\tau}_N^H(s_t) \to \hat{z}_{s_t} \to a_t \to s_{t+1} \to \dots \tag{6}$$

which will be recorded as our (longer-context) training data:

$$\mathcal{D}_{\text{SKL-RL}} \leftarrow [s_0, (\boldsymbol{\tau}_N^H(s_0), \hat{z}_{s_0}, a_0), s_1, (\boldsymbol{\tau}_N^H(s_1), \hat{z}_{s_1}, a_1), s_2..., s_T, R] \tag{7}$$

**III. Outcome-Driven Verification**: Unlike the rejection sampling in SKL-SD, the correctness of the bootstrapping step is implicitly supervised by the final reward $R$. We optimize these interconnected bootstrapping loops as a long model response using GRPO.

**IV. Knowledge Internalization** : To consolidate the simulation-derived insights into the agent's parametric memory, we maintain a distillation dataset $\mathcal{D}_{\text{distill}} = \{(z_{s_t}, \hat{z}_{s_t})\}_{s_t \in \mathcal{S}}$ that pairs the prior stateful knowledge $z_{s_t}$ with its refined, post-simulation counterpart $\hat{z}_{s_t}$. This optimization step enables the agent to internalize look-ahead foresight, allowing it to build upon the updated knowledge $\hat{z}_{s_t}$ to guide rollouts in the subsequent iteration as:

$$\cdots \to s_t \to \boldsymbol{\tau}_H^N(s_t|z_{s_t} = \hat{z}_{s_t}) \to \hat{z}'_{s_t} \to a'_t \to s'_{t+1} \to \dots \tag{8}$$

Further details regarding the multi-turn distillation mechanics are provided in Appendix I. The joint optimization objective for SKL-RL with coefficient $\lambda$ is formulated as:

$$\mathcal{L}_{\text{SKL-RL}} = \mathcal{L}_{\text{GRPO}}(\mathcal{D}_{\text{SKL-RL}}) + \lambda \cdot \mathcal{L}_{\text{Distill}}(\{(z_s, \hat{z}_s)\}_{s \in \mathcal{S}}) \tag{9}$$

Intuitively, SKL-RL drives the agent to verify its stateful knowledge hypothesis $z_{s_t}$ through active simulation $\boldsymbol{\tau}$, extracting insights from these localized experiences to ground an exploitative action for a specific state. The inclusion of the distillation objective is intended to amortize part of the computational cost of the look-ahead search and bootstrapping loop into a direct inference step. In the current ChessPuzzles study, however, a separate zero-simulation evaluation does not show a noticeable improvement, as reported in Section 4.2.3. In contrast to **SKL-SD**, which depends on heavily orchestrated data curation and verification pipelines, **SKL-RL** provides a fully autonomous learning paradigm. By optimizing directly from sparse environmental rewards, it bypasses the bottleneck of manual data curation, at the cost of increased training complexities due to long-context RL.

Crucially, when the simulation traces $\boldsymbol{\tau}_N^H$ are also self-generated through multi-turn rollouts, SKL-RL uniquely couples exploration (via simulation) and exploitation (via knowledge bootstrapping). Optimizing the entire interaction trajectory via $\mathcal{L}_{\text{GRPO}}$ induces an emergent, **self-adaptive balance**: excessive exploration introduces behavioural noise that destabilizes the bootstrapping signal, whereas overly conservative simulation limits the agent's foresight, leading to myopia. Our chess reasoning evaluation (Section 4.2) empirically confirms this emergent dynamic: the agent attempts to resolve this tension during end-to-end RL training.

## G Compare SKL-SD and SKL-RL to other training methods

The **Stateful Knowledge Learning** framework aligns with the *Learning from Language Feedback* (Cheng et al., 2023) paradigm, as our bootstrapping loop is essentially updating state knowledge

using evaluative language generated at subsequent states. The central insight is that **language feedback conveys richer semantic information than scalar rewards** (Feng et al., 2024), regardless of whether it comes from external environments, self-reflection, other models, or human supervision. Within this paradigm, we here compare our methods to recent training-based methods designed to improve self-reflection capabilities.

Critic-GRPO (Zhang et al., 2025b), Reflect-Retry-Reward (Bensal et al., 2025), and R$^3$L (Shi et al., 2026b) explore how evaluative behaviour evolves under reinforcement learning. These approaches construct datasets consisting of initial experiences $\mathcal{D}_{\text{base}}$, self-generated reflections $\{d_{\text{reflection}}\}$, and refined experiences $\mathcal{D}_{\text{new}}$ (trajectory-level) or retried experiences $\mathcal{D}_{\text{retry}}$ (step-level), followed by RL training on the combined dataset $\mathcal{D} = \{\mathcal{D}_{\text{base}}, d_{\text{reflect}}, \mathcal{D}_{\text{new/retry}}\}$. They also synthesize reflective reasoning paths by concatenating base-level exploration traces with filtered refinement, denoted as $\{\mathcal{D}^{\text{distill}}_{\text{reflect}}, \mathcal{D}^{\text{distill}}_{\text{retry}}\}$. Table 4 lists the typical training objectives (policy gradient losses) used for different components. Critic-GRPO (Zhang et al., 2025b) can be seen as reinforcing with objectives 1&2. Reflect-Retry-Reward (Bensal et al., 2025) reinforces on objective 6, R$^3$L (Shi et al., 2026b) is trained with RL on objective 1&3, and perform SFT on objective 4&5. Experiential RL (Shi et al., 2026a) optimize with respect to objective 2&4.

In our work, **State-Knowledge-Learning Self-Distillation** (see Equation 4) adopts a similar loss design, but all components follow a *stateful* formulation where state knowledge $z$ is generated before the action $a$. Specifically, we maintain a policy gradient loss over base rollouts $\mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{base}})$ (analogous to objective 1, but with stateful trajectories and sample filtering), a policy gradient loss over distilled refined responses $\mathcal{L}_{\text{PG}}(\mathcal{D}_{\text{distill}})$ (analogous to objective 2, again in a stateful form and filtered), and an auxiliary SFT loss $\mathcal{L}_{\text{SFT}}(\mathcal{D}_{\text{reflect}})$ that guides the bootstrapping behaviour. Thus, **SKL-SD can be viewed as a "stateful" version of recent self-reflection RL training methods**; we call it "self-distillation" or "self-imitation" because we reinforce only filtered (positive) trajectories and discard negative samples.

In contrast, **State-Knowledge-Learning Reinforcement Learning** (see Equation 9) is **fundamentally different**: it integrates state aggregation and the bootstrapping loop into a single online rollout episode and relies solely on the final outcome reward to jointly shape both generation and evaluation behavior. This makes it a more unified and autonomous agent training framework.

| **Objective** | **Loss Function** (with or without sample filtering) |
|---|---|
| 1. Base Response | $\mathbb{E}_{\mathcal{D}_{\text{base}}}\left[R_{\text{base}} \cdot \nabla_\theta \log \pi_\theta(\tau_{\text{base}} \mid s_0)\right]$ |
| 2. Distilled Refined Response | $\mathbb{E}_{\mathcal{D}^{\text{distill}}_{\text{reflect}}}\left[R_{\text{reflect}} \cdot \nabla_\theta \log \pi_\theta(\tau_{\text{new}} \mid s_0)\right]$ |
| 3. Distilled Retried Response | $\mathbb{E}_{\mathcal{D}^{\text{distill}}_{\text{retry}}}\left[R_{\text{retried}} \cdot \nabla_\theta \log \pi_\theta(\tau_{\text{base},<\text{pivot}}, \tau_{\text{retry},>\text{pivot}} \mid s_0)\right]$ |
| 4. Complete Refinement Process | $\mathbb{E}_{\mathcal{D}_{\text{reflect}}}\left[R_{\text{reflect}} \cdot \nabla_\theta \log \pi_\theta([\tau_{\text{base}}, d_{\text{reflection}}, \tau_{\text{new}}] \mid s_0)\right]$ |
| 5. Complete Retry Process | $\mathbb{E}_{\mathcal{D}_{\text{retry}}}\left[R_{\text{retry}} \cdot \nabla_\theta \log \pi_\theta([\tau_{\text{base}}, d_{\text{reflection}}, [\tau_{\text{base},<\text{pivot}}, \tau_{\text{retry},>\text{pivot}}]] \mid s_0)\right]$ |
| 6. Reflection | $\mathbb{E}_{\{d_{\text{reflection}}\}}\left[R_{\text{reflect}} \cdot \nabla_\theta \log \pi_\theta(d_{\text{reflection}} \mid s_t, \tau_{\text{base}})\right]$ |

Table 4: Training objective for different purposes in works related to other self-reflection training methods.

## H LOCALIZED PARALLEL SIMULATION IN SKL-RL

In the context of SKL-RL, the localized parallel simulation mechanism acts as an online, text-based look-ahead tree exploration. Because the agent has access to a resettable simulator, it doesn't just pick an action and move forward blindly; instead, it pauses at the current state $s_t$, freezes the main trajectory, and "probes" the future. Here is a detailed breakdown of how this parallel simulation operates at each timestep:

1. **State Anchoring** (The Root): At any given online step, the agent encounters state $s_t$. The simulator's state is checkpointed or saved at this exact position, acting as the root node for the simulations.

2. **Multi-Branch Spawning** (Breadth $N$): From this single root state $s_t$, the agent generates knowledge $z_{s_t}$, and instantiates $N$ actions for independent, parallel branches (or simulation workers).
3. **Independent Temporal Expansion** (Horizon $H$): At each subsequent simulation timestep $1, \ldots, H$, every branch rolls out independently. Each branch samples knowledge and actions according to the agent's current policy (or an exploration policy) and receives independent observations and rewards from its own isolated instance of the environment.
4. **Experience Aggregation**: Once all $N$ branches reach the maximum simulation horizon $H$ (or hit a terminal state), their complete rollout histories are gathered into an aggregated experience block $\boldsymbol{\tau}_N^H$.

A concrete example of parallel simulation on ChessPuzzles is provided in Figure 10. Figure 9 also shows an experiment of how different budgets $(H, N)$ affect performance on ChessPuzzles.

# I MULTI-TURN SELF-DISTILLATION IN SKL-RL

The primary objective of the distillation loss $\mathcal{L}_{\text{Distill}}(\{z_s, \hat{z}_s\}_{s\in\mathcal{S}})$ is to internalize the insights gained from look-ahead simulations into the agent's prior stateful knowledge. By grounding its reasoning in previously distilled knowledge, the agent can more effectively explore alternative trajectories in subsequent iterations and incrementally compound its predictive understanding. Formally, this iterative refinement process across consecutive iterations can be expressed as:

$$\text{Iteration } i :... \to s_t \to \{z_{s_t}, a_t, s_{t+1}, z_{s_{t+1}}, ..., s_{t+H}\} \to \underline{\hat{z}_{s_t}} \to a_t \to s_{t+1} \to ...$$
$$\text{Iteration } i+1 :... \to s_t \to \{\underline{\hat{z}_{s_t}}, a_t^{'}, s_{t+1}^{'}, z_{s_{t+1}^{'}}, ..., s_{t+H}^{'}\} \to \hat{z}_{s_t}^{'} \to a_t^{'} \to s_{t+1}^{'} \to ...$$

This process constitutes a teacher-student framework. The "student" model represents the agent's zero-shot capability to generate stateful knowledge upon first encountering a state, governed by the distribution $\pi(z_{s_t}|s_t, \mathcal{H})$, where $\mathcal{H}$ denotes the interaction history context. The "teacher" model represents the target distribution $\pi(\hat{z}_{s_t}|\boldsymbol{\tau}_N^H, s_t, \mathcal{H})$, which produces the refined stateful knowledge conditioned on the outcomes of the look-ahead simulations $\boldsymbol{\tau}_N^H$.The goal of this self-distillation is to align the student model with the teacher, compelling the agent to consolidate simulation-derived experience and autonomously build new knowledge. This alignment can be optimized using either Forward or Reverse Kullback-Leibler (KL) divergence in a multi-turn setting:

$$\textbf{Forward KL} : \mathcal{L}_{\text{Distill}}^{\text{Foward}}(\theta) = \sum_{s_t} \text{KL}\Big[\text{stopgrad}\big(\pi_\theta(z|\boldsymbol{\tau}_N^H, s_t, \mathcal{H})\big) \Big\| \pi_\theta(z|s_t, \mathcal{H})\Big] \tag{10}$$

$$\textbf{Reverse KL} : \mathcal{L}_{\text{Distill}}^{\text{Reverse}}(\theta) = \sum_{s_t} \text{KL}\Big[\pi_\theta(z|s_t, \mathcal{H}) \Big\| \text{stopgrad}\big(\pi_\theta(z|\boldsymbol{\tau}_N^H, s_t, \mathcal{H})\big)\Big] \tag{11}$$

Here, the Forward KL objective serves as a multi-turn, self-distillation extension of Supervised-KD (Agarwal et al., 2024), while the Reverse KL acts as a multi-turn extension of OPSD (Zhao et al., 2026).

# J EXPERIMENTS ON AGENTIC ENVIRONMENTS

## J.1 EXPERIMENT SETUP

The experimental setup of SKL-SD on agentic tasks is highly aligned with the setup used by (Shi et al., 2026b) as we re-implement SKL-SD in their public codebase, and use the hyperparameter setting reported in their paper. Extra hyperparameter introduced includes bootstrapping budget $(H = 3, N = 3)$ and loss coefficients $\lambda_1 = \lambda_2 = 1$. The baseline includes rejection sampling methods like RAFT (Dong et al., 2023), GRPO variants including the default GRPO (Guo et al., 2025a), OPMD (Yao et al., 2025), and GSPO (Zheng et al., 2025). Crucially, we compare against language-feedback frameworks that utilize self-reflection or critiques, including Critique-GRPO (Zhang et al., 2025b), Reflect-GRPO (Bensal et al., 2025), and R$^3$L (Shi et al., 2026b). All experiments

are implemented using the Trinity-RFT framework (Pan et al., 2025) to ensure parity in infrastructure. We use *Qwen2.5-7B-Instruct* (Yang et al., 2024) as the backbone model across all methods. Training is conducted on an NVIDIA H100 single-node cluster.

The environment also follows their setups. The state in WebShop contains the textual description of the current webpage. The state in ScienceWorld contains the limited observations at different game statuses. We use the same system prompt as in (Shi et al., 2026b) and the prompt for bootstrapping is provided in Table 5.

Table 5: Prompt for bootstrapping

You will be given the history of a past experience starting at the same root state, in which you were trying to solve the task but failed. You might not see the final outcome in each trajectory, but you will see the understanding of the final state in each trajectory. Take it into your consideration.

You will also be given your previous understanding of the root state. Based on the history, update your understanding of the root state.

### J.2 Detailed reasoning trace

Table 6: A reasoning example on WebShop. Multiple state knowledge revealed at different states

**Query**:
Find me non-slip, easy clean computer armoires with PU leather with color: white, and size: 120x60cm, and price lower than 70.00 dollars

---

**[State Knowledge for Initial Search]**
**State Understanding**:
1. Search Queries:
   - Ensure the search terms include specific attributes and measurements to match the exact product requirements: - Color: ... - Size: ... - Additional Features:... - Price: ...
2. Product Verification and Filtering:
   - After locating a potential product, verify all criteria (color, size, features, price) are correctly specified.
   - Directly click "Buy Now" if the product matches all specified criteria, without additional checks for availability.
3. Buy Now on Detailed Product Page:
   - Ensure that the click on the "Buy Now" button is performed directly on the detailed product page with all filters correctly applied.
   - Click the "Buy Now" button if all conditions are verified.
4. Detailed Review and Double-Check:
   - Verify that the product's page correctly displays all the required attributes and features, using inspection tools if necessary.
   - Double-check that the product's details, especially dimensions (size: 120x60cm) and color (white), and price ($\leq$ \$70.00) match the criteria.
5. Error Logging:
   - If a product does not match all criteria, log the relevant details of the product and try again, ensuring a revisit to the search page with corrected parameters if necessary.

**Steps for Implementation:**
1. Refine Search Query with Specific Attributes and Dimensions: .....

---

**[State Knowledge for Browsing]**
**State Understanding:**
1. Correct Product Matching: Ensure that the selected product details (color, size, price)—all attributes must correctly match the required criteria: 1. Color: ..., 2. Size: ..., 3. Price: ...
2. Product Detail Verification:

- Double-check each product detail on the result page to ensure accuracy.
- Verify that images and descriptions explicitly confirm the exact details being used (white color, 120x60cm size, etc).

.....
N. Notes for Future Runs:

- Cross-Verify Attributes: Validate the exact color, size, and price on each detailed product page.
- Double-check Final Product: If unsure, re-verify the product details and attributes before clicking Buy Now.
- Snapshots as Evidence: Take screenshots or notes of the product description and attributes to confirm they meet the criteria.

---

**[State Knowledge for Specific Product Page]**
**State Understanding**:
1. Navigating the URL: Start by ensuring that the correct URL is accessed, specifically B095KCTB3Y, to display the desired product.
2. Selecting Size and Color Details:

- Ensure that the size 120x60cm and color white are explicitly selected and chosen correctly from dropdown options (if available).
- Double-check that the product details on the page reflect the correct options and attributes (white, 120x60cm, non-slip).

3. Direct Purchase Action: Once all conditions (size and color) are correctly matched and verified on the product description, click the Buy Now button.
**Notes**:

- Ensure that the search query uses keywords such as "white", "120x60cm", and "pu leather" to precisely target the correct product.
- Directly interact with the specified URL and verify the exact attributes before finalizing the purchase.
- Use the correct Buy Now button to conclude the task.
- For future iterations, focus on refining the search criteria to eliminate confusion.

# K EXPERIMENTS ON CHESSPUZZLES

## K.1 THE UNIQUENESS OF THE CHESS GAME FOR LLM EVALUATION

Recent research on LLM-based agents often evaluates performance on relatively simple agentic environments, such as ALFWorld (Shridhar et al., 2020), WebShop (Yao et al., 2022a), or grid-based games like FrozenLake and Sokoban (Feng et al., 2024). While these benchmarks are useful for studying sequential decision-making and tool-use behaviors, they typically involve limited strategic depth and can often be solved through short-horizon reasoning or heuristic exploration. Consequently, many agent studies primarily report final task completion rates, without closely examining the soundness or internal consistency of the reasoning traces that lead to those outcomes. In contrast, chess games require substantially richer strategic reasoning (Liu et al., 2025). Solving a chess puzzle generally involves multi-step planning, evaluation of alternative move sequences, and understanding tactical motifs such as checks, pins, or forced mates. These characteristics make chess a more demanding testbed for assessing whether an agent can produce coherent reasoning processes rather than merely reaching the correct final answer.

Another advantage of chess puzzles is that they are less susceptible to data contamination compared to many popular agent benchmarks. Many environments have well-documented task templates that may appear in pretraining corpora, making it difficult to disentangle memorization from genuine reasoning. In contrast, our experiments (Figure 9) indicate that even strong closed-source models (e.g., GPT-5) struggle to solve the ChessPuzzles benchmark in a zero-shot setting, as also illustrated

in Table 7. This suggests that the model cannot simply recall solutions from training data but must instead reason about the game dynamics.

Taken together, these properties make chess puzzles an ideal evaluation setting for studying an agent's ability to adapt to an environment with minimal prior knowledge of the underlying rules. This motivates our use of chess as a challenging benchmark to evaluate the effectiveness of our algorithm.

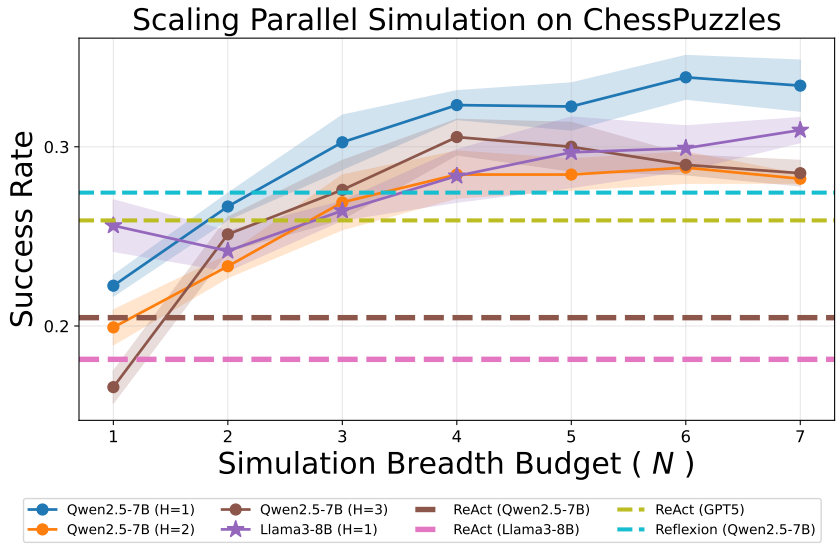



Figure 9: Performance of SKL-RL inference with different parallel simulation budgets $(H, N)$ without fine-tuning. We evaluate on Qwen2.5-7B-Instruct and Llama-3-8B-Instruct, comparing against baselines such as ReAct and Reflexion. Notably, ReAct utilizing GPT-5 as the backbone model exhibits poor zero-shot performance on the ChessPuzzle task.

Table 7: Reasoning of based model in ChessPuzzles tasks. Both Qwen2.5-7B-Instruct and GPT-5 fail to provide good reasoning.

**Game State: 1Qr5/p4r1k/8/3p2p1/P2P2P1/2P2pKP/8/8 w - - 0 49 , StockFish's Move: Qb1**

**Qwen2.5-7B-Instruct (Nonsensical reasoning)**

Let's analyze the position strategically. White has a queen and a king, while Black has a rook and a king. The position is closed, with few pieces on the board, so tactical play is important.

**Key candidate moves to consider are**:
1. Qxc8: Captures the pawn and puts pressure on the black king. **(Misidentified the piece)**
2. Qa8: Attacks the rook on a1 and also threatens the black king. **(Misidentified the piece)**
3. Qc7: Targets the black king directly. **(Wrong justification)**

Now let's **evaluate each move**: **(Completely wrong)**
1. **Qxc8**: If this move is played, Black will respond with Rxa8. The resulting position will leave White with a potential fork with the queen, targeting both the black king and the rook. However, this might be risky as Black could still create threats with the rook on a8.
2. **Qa8**: This move attacks the rook on a1 and threatens to put pressure on the black king. After Qa8, Black can respond with Rxa8, but this would result in a simplified position where White can focus on controlling the center and potentially queening the pawn on h4 or a5.
3. **Qc7**: This move directly targets the black king, putting it under pressure. If Qc7 is played, Black can respond with Kf8, which limits the immediate threat. However, it leaves the black king more exposed.

Considering the strategic implications, Qa8 seems to be the most promising move. It not only attacks a key piece but also creates a significant threat to the black king. The resulting position is simpler and allows White to control the center and potentially create winning chances. `<move>Qa8</move>`

**GPT-5 (Better at general reasoning, but fail at detailed analysis)**

**Strategic evaluation (Making Sense)**:

Material is heavily in White's favor because White has a queen (on b8) against Black's rook (on f7) plus some pawns. The immediate priority is to simplify into a trivially winning endgame by trading queens/major pieces or winning the opposing rook,

while also keeping White's king safe. Black's king is on h7 and is relatively exposed; Black's main counterplay is the rook on f7 plus the advanced f3-pawn.

**Three key candidate moves**:

1) Qxc8
Idea: capture the black queen on c8 and trade queens immediately.
Likely response: Black must recapture with Rxc8 (the rook from f8 goes to c8).
Outcome: White has traded queens, but then White is only up material modestly and still has to deal with Black's active rook and passed f-pawn. This is good and likely winning, but it gives Black some practical chances compared to winning the rook outright. **(Wrong detailed analysis)**

2) Qxa7
Idea: win a pawn with check possibilities later (queen near the king).
Likely response: Black can continue with f2 (or Rxa7 isn't possible), pushing the passed pawn and creating real promotion threats, while White hasn't reduced Black's major-piece activity.
Outcome: This is slower and allows Black counterplay with the f-pawn and rook coordination. Not the cleanest conversion.

3) Qb7
Idea: directly attack Black's rook on f7 along the 7th rank and win it.
Likely response: Black's rook has no squares available on the 7th rank because b7 controls the entire rank; it also cannot capture Qb7 (rook on f7 can't move like that) **(Wrong prediction)**, and moving the king doesn't solve the attack. The only "defense" would be moving the rook away—but from f7, every legal rook move is along the f-file or 7th rank, and Qb7 covers f7 and the whole 7th, so the rook is effectively lost.
Outcome: White wins the rook cleanly, after which the position is completely winning with queen versus only pawns.

**Conclusion**:
Qb7 is the most forcing and highest-value move: it wins Black's rook immediately with minimal counterplay, converting the advantage far more cleanly than trading queens with Qxc8 or grabbing a pawn with Qxa7. `<move>Qb7</move>`

### K.2 Experiment Setup

**Experiment Setup** We also compare SKL-RL with other training baselines introduced in Section 4.1, all of which operate via trajectory-level self-reflection. All methods are trained with matched computation budgets, measured by the number of LLM calls during training. We use Qwen2.5-7B-Instruct as the base model, and apply GRPO for all RL implementations. We select a subset of the original ChessPuzzles (rating $< 1000$) and split it into a train and a test set.

On ChessPuzzles, each state $s \in \mathcal{S}$ includes a FEN notation that describes the chessboard position. Agents are prompted to generate state knowledge $z(s_t)$ for the current game state $s_t$ and decide on the movement $a_t$ to make in two consecutive reasoning steps:

1. **State Knowledge** $z_t$: `[User]: What is your understanding of the game state: {state}. [Assistant]: ...`
2. **Movement** $a_t$: `[User]: You are in stage {stage}, based on the state understanding, what is the decided move? [Assistant]: ...`

where the state knowledge explicitly queries the model for its internal predictive knowledge of the game state, and the movement query will use the generated understanding as the context, asking the model to directly generate action at each stage (i.e., both during simulation or bootstrapping). This way of reasoning can also avoid the issue of overthinking that causes missing action tokens and make knowledge distillation easier to implement later. Figure 10 illustrates this. Self-generated tree search simulation is carried out in a similar way as parallel decoding, where the agent is prompted to generate multiple ($N$) moves at the same time, and for each rollout branch, the agent independently performs a single-trajectory rollout until running out of horizon budget $H$. Table 8 shows the system prompt for ChessPuzzles.

Table 8: System Prompt used in ChessPuzzles tasks.

A conversation between User and Assistant.
The User asks the best move to make for a given chess board state, and the Assistant solves it.
The Assistant is a professional chess player who can plan a few steps ahead by interacting with a simulator, and then think about the reasoning process and at last provides the User with the answer.

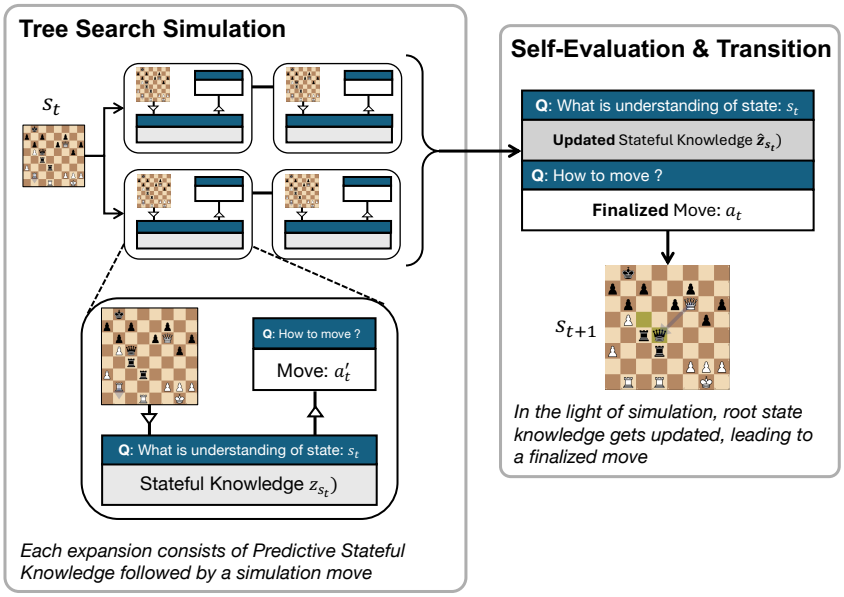



Figure 10: At each state $s_t$, the agent first performs a tree search where each node expansion consists of state knowledge and a simulation move. The generated simulation experience $\boldsymbol{\tau}_N^H(s_t)$ is aggregated as the context, and the agent summarizes an updated state understanding $\hat{z}_{s_t} \sim \pi_\theta(\cdot|\boldsymbol{\tau}_N^H(s_t), s_t, z_{s_t})$ as well as a finalized move $a_t$ to transition to the next state $s_{t+1}$.

The Assistant's thinking process must include an understanding of the game state, based on which the Assistant can later decide on the next move.
The understanding should describe how the Assistant analyzes the current position, including:
- A strategic evaluation of the position.
- A comparison of key candidate moves (not all of them) in terms of their possible subsequent outcomes.
- Do not try to enumerate all possible moves and outcomes, only consider the most likely ones.

The decision on next move must be in SAN notation, strictly using the moving piece and the destination square (e.g., Nf3, Rxf2, c5).

Reminder of chess rules:
- Bishops move diagonally.
- Rooks move horizontally or vertically.
- Knights jump in an L-shape.
- Queens combine rook and bishop movement.
- Kings move one square in any direction.
- Pawns move forward, capture diagonally, and can promote.

### K.3 Ablation on bootstrapping behaviour at different heuristic simulation strategies

Table 9: Reasoning trace example of ablation study for SKL-RL in Section 4.2.1.

**(Test set) Game State: 2r3k1/p1r2ppp/2N1p3/3pPP2/4b1P1/1P6/P6P/2R2RK1 w - - 1 28**

**Simulation Strategy: Expert Move Rate** $\frac{1}{2}$ in simulation **(short, frequency-based)**

The given game state is: 2r3k1/p1r2ppp/2N1p3/3pPP2/4b1P1/1P6/P6P/2R2RK1 w - - 1 28. From the simulation trajectories, we can see that the move 'Nd8' is the most frequently chosen move by the player, and it leads to a few different successor states. Let's analyze the trajectories:

[Simulation Trajectory]

From the trajectories, we can see that the move ‘Nd8‘ leads to the same or similar states in all cases where it is played. On the other hand, the move ‘Ne7‘ leads to a different state. The state after ‘Nd8‘ is: 2rN3k/p1r2ppp/4p3/3pPP2/4b1P1/1P6/P6P/2R2RK1
This state is very similar to the original state, suggesting that ‘Nd8‘ is a good move as it maintains the position quality. The state after ‘Ne7‘ is: 2r4k/p1r1Nppp/4p3/3pPP2/4b1P1/1P6/P6P/2R2RK1
This state is different and potentially less favorable, as it involves a knight move that could be less effective in maintaining the position.

**Strategic evaluation**:

- The position is balanced, with both sides having similar material and pawn structures.
- The move ‘Nd8‘ supports the king and maintains the structure, while ‘Ne7‘ leads to a different and potentially less favorable position.

Based on the simulation rollouts and the strategic evaluation, the best move is: **Nd8**

---

## Simulation Strategy: Expert Move Rate $\frac{1}{6}$ in simulation (more structured, but hallucinate)

The given state is: 2r3k1/p1r2ppp2N1p3/3pPP24b1P1/1P6P6P/2R2RK1w - -1 28. This is a middlegame position with White to move. Let’s analyze the key features of this position:

1. **Material Balance**: The material is balanced with two rooks (R), two knights (N), and one bishop (B) for each side. White has a pawn (P) on the 8th rank (White’s king-side), and Black has a pawn on the 7th rank (Black’s king-side).
2. **King Safety**: Both kings are castled on the kingside, and neither side has any immediate threats to their kings.
3. **Pawn Structure**: White has a pawn on e2 and pawns on f2 and g2. Black has pawns on e7 and f7.
4. **Active Pieces**: White has a knight on b8 and a rook on the 1st rank. Black has a knight on a5 and a rook on the 1st rank.
5. **Key Moves**: The knight on b8 (Nb8) in Traj 3 suggests that it might be a strong move to support the pawn structure and potentially create threats. The knight moves Ne7 and Nb5 in Traj 1 and 2 suggest support for the king-side pawns and maintaining control over key squares. The rook move Na5 in Traj 4 suggests a support move for the knight and maintaining control over key files.

**Analysis of Moves**

- **Move: Nb8 (Traj 3)**: Supports the pawn structure, maintains control over key squares, and potentially prepares for future pawn moves.
- **Move: Ne7 (Traj 1)**: Supports the king-side pawns and maintains control over key squares.
- **Move: Na5 (Traj 4)**: Supports the knight and maintains control over key files.

**Conclusion**: The move Nb8 in Traj 3 seems to be a strong candidate as it supports the pawn structure and maintains control over key squares. Given the similarity in the trajectories, maintaining support and control over key squares appears to be a strong strategy.
**Recommendation**: The best move for White in the given state is: **Nb8**

---

## Simulation Strategy: Random Move in simulation (ignore simulation, more explicit reasoning but severe hallucination)

**Analysis of the Game State**
The root state of interest is: 2r3k1/p1r2ppp/2N1p3/3pPP2/4b1P1/1P6/P6P/2R2RK1 w - - 1 28.
This state is derived from the four trajectories provided. Let’s break down the key elements of this position:

1. **White to Move (w - - 1 28)**: White is to move, and it is the 28th move of the game.
2. **Material Balance**: White has 2 rooks, 3 knights, 1 king, and 1 pawn. Black has 2 rooks, 1 knight, 1 king, and 1 pawn.
3. **Positional Evaluation**: White has a rook and a knight, while Black has a rook and a knight. White has a king and a rook, while Black has a king and a rook. White has a pawn on the 6th rank, while Black has a pawn on the 6th rank.
4. **Key Pieces**: White’s rook on d1 and Black’s rook on c4. White’s knight on g5 and Black’s knight on d8.
5. **Active Play**: The game is still in the middlegame, with no clear endgame or decisive advantage. Both sides have active pieces, and the game is likely to continue with tactical exchanges and positional maneuvers.

**Trajectory Analysis**: Let’s analyze the trajectories to understand the potential future value of the root state:

- **Traj 1, 2, 3, 4**: The state remains similar, with no clear changes in the position.

**Strategic Evaluation**

- **Control of the Board**: Both sides have active pieces, and the board is open.
- **King Safety**: Both kings are relatively safe, but the rooks and knights are more active.
- **Pawn Structure**: The pawns on the 6th rank are key, and both sides have similar pawn structures.

**Candidate Moves**: Given the current state, let’s consider the most likely candidate moves:

1. **g5**: This move might aim to control the g5 square and potentially create a tactical opportunity.
2. **h4**: This move might aim to control the h4 square and potentially create a tactical opportunity.
3. **Rfd1**: This move might aim to control the d1 square and potentially create a tactical opportunity.

**Conclusion**: Based on the analysis of the trajectories, the root state is still in a balanced middlegame position with active pieces. The moves g5, h4, and Rfd1 are all plausible and aim to create tactical opportunities or control key squares.
**Recommended Move**: Given the strategic evaluation and the potential future value from the trajectories, the move **g5** seems to be a strong candidate as it controls a key square and potentially creates tactical opportunities. **Recommended Move: g5**

### K.4 A COMPLETE EXAMPLE OF SKL-RL WITH SELF-GENERATED SIMULATION

Table 10: Complete reasoning trace of SKL-RL at state $s_t$.

**[System Prompt]**:
(See Table 8)

**[User]** **(Environment State $\mathbf{s}_t^E$)**
Current Game State:
`k1r4r/p4pp1/2Nb2p1/3P4/Q3p3/3nPq1P/P2K1P2/R4R2 w - - 4 29`

Root State Simulation $s_t$

**[User]** **(Root State $\mathbf{s}_t = s_t^E$ in Simulation)**
Current Simulation State: `k1r4r/p4pp1/2Nb2p1/3P4/Q3p3/3nPq1P/P2K1P2/R4R2 w - - 4 29`

What is your understanding of the game state:
`k1r4r/p4pp1/2Nb2p1/3P4/Q3p3/3nPq1P/P2K1P2/R4R2 w - - 4 29`

**[Assistant]** **(Understanding $z_{s_t}$ at Root State)**
**### Understanding of the Game State**
The current position is . . .

**[User]**
You are now in simulation, your current goal is to move in order to better understand potential future outcomes of the game, so you can either explore or exploit. You can generate at most 3 **(budget $N$)** candidate moves to be tested with the simulator. Do not generate duplicate moves.
Choose the move from the available move list: . . .

**[Assistant]** **(Simulation Action $a_t^{(i)}$ at Root State)**
`<move>Qxa7</move> <move>Qa6</move> <move>Ne7</move>`

**[User]** Found 3 valid moves. **(Next State $s_{t+1}^{(i)}$ in Simulation)**
- **Traj 1**: Subsequent result of move Qxa7: Results: You have successfully solved the game within this simulation!
- **Traj 2**: . . . . . .
- **Traj 3**: . . . . . .

Expanded State Simulation $s_{t+1}$

**[User]** & **[Assistant]** **(Continue Simulation at $s_{t+1}^{(i)}$)**
(Turn 2 simulation) . . .

Self-Evaluation/Bootstrapping for $\hat{z}(s_t)$ and $a_t$

**[User]** **(Simulation Ends, Start Bootstrapping and Get $\hat{z}_{s_t}$)**
[Summarizing stage]
The simulation ends. Lets pause, check the interaction history, and rethink again both the benefits and harm of each possible moves at simulation step 1.

**### Simulation History**: . . .

**[User]**
Based on the simulation history, what is your updated understanding of the game state: `k1r4r/p4pp1/2Nb2p1/3P4/Q3p3/3nPq1P/P2K1P2/R4R2 w - - 4 29`

**[Assistant]** **(Finalized action $a_t$ at $s_t$)**
**### Updated Understanding of the Game State** . . .

**[User]**
The simulation ends, your current goal is to output the optimal move for the root state. Based on previous simulation, your understanding of the game state and your current goal, what is the best move to make?

**[Assistant]**
<answer>Qxa7</answer>

### K.5 Reasoning trace for R$^3$L and SKL-RL with self-generated simulation after end-to-end training

Table 11: Reasoning trace example ONE of SKL-RL after end-to-end RL training. SKL-RL helps with more concise, stateful, outcome-aware reasoning.

**Game State**:
r1b1k2r/5ppp/2p5/p1q1p1B1/Pp2P1n1/1P3NN1/1P3PPP/3Q1RK1 w kq - 0 17

---

**[State Understanding]**

The current position is a complex middlegame position for White, who is in a strong position due to their queen and rook being active and threatening the black king. White has a strong initiative and should focus on developing their queenside attack while also preparing to push their queenside pawns.

Let's consider the most promising moves:
1. **Qd8**: This move is very strong because it not only attacks the black queen but also threatens to checkmate on the next move if the queen moves. **(Correctly identify the key benefits)**
2. **Kh1**: This move is more positional and prepares to push the h-pawns, which could be a long-term strategy.
3. **Re1**: This move is more defensive but also prepares to support the h-pawns and control the e-file.
4. **Nxe5**: This move is more tactical and aims to put pressure on the black knight and potentially open up the e5 square for further attacks.

**(analysis seems reasonable but still at a surface-level)**

Given the strong positional and tactical threats, the move Qd8 stands out as the most aggressive and decisive.

---

**[Experience Summary/Bootstrapping]**

Based on the simulation history, the updated understanding of the game state is as follows:
1. White is in a strong position, having just won a queen and a rook, which is a significant material advantage. **(wrong)**
2. White has a strong attack on the black king, with the queen and rook threatening to checkmate on the next move. **(correctly summarize the key benefits)**
3. The black king is in a vulnerable position, with the queen and rook threatening to deliver checkmate.
4. The black queen is a potential threat, but it is not well-placed to counter the white attack.
5. The white king is safe and can continue to support the attack on the black king.
6. The board is open and dynamic, with many pieces still active and ready to participate in the attack.

Given this updated understanding, the position is a strong position for White, and the move **Qd8** was the best move in the simulation history, leading to a win. **(combine knowledge with simulation result)**

Table 12: Reasoning trace example of R$^3$L. Trajectory reflection from R$^3$L ignores stateful information.

**Game State**:
r1b1k2r/5ppp/2p5/p1q1p1B1/Pp2P1n1/1P3NN1/1P3PPP/3Q1RK1 w kq - 0 17

---

**[Trajectory Reflection]**

**trajectory summary**: The Assistant analyzed the chess position and decided on a sequence of moves focusing on gaining positional advantage, promoting pawns, and controlling key squares. However, the Assistant failed to complete the task within the step limit, as no pawn promotion occurred despite several opportunities.

**root cause analysis**: Why was the approach suboptimal? Because the Assistant did not prioritize pawn promotion in the key moments. Why did the Assistant not promote pawns? Because the strategy was too positional and did not explicitly plan for pawn promotion. Why this positional approach? Because the Assistant optimized for immediate positional advantages rather than focusing on the critical goal of promoting pawns. Root cause: Lack of explicit focus on the endgame goal of promoting pawns. The Assistant should have immediately promoted a pawn when given the opportunity, as the game was nearing its end.

**trajectory outcome**: failure,

**improvement suggestion**: In endgame scenarios, especially in rook and pawn situations, the primary goal should be to promote pawns. The Assistant must explicitly prioritize pawn promotion and take advantage of opportunities to do so, even if it means sacrificing other positional benefits. For instance, when the King is closer to the fifth rank (more pawn promotion squares), the Assistant should move towards those squares and promote the pawn as soon as possible. Example: In the current trajectory, when the white king was on g7 or h6, the priority should have been to move towards a pawn promotion square and promote the pawn immediately when legal.

**retry from step**: 0
**(Coarse-grained Understanding)**

Table 13: Reasoning trace example TWO of SKL-RL before and after end-to-end RL training.

**Game State: 2r2k2/1K1r1p2/5Q2/pp2pN2/P3P3/2P4P/1P3q2/8 w - - 3 46; Optimal Move: Kxc8**

**Experience Summary/Bootstrapping before RL**

The current game state is as follows: 2r2k21K1r1p25Q2pp2pN2P3P32P4P1P3q28

**Updated Understanding of the Game State**: 2r2k21K1r1p25Q2pp2pN2P3P32P4P1P3q28

**White to move.**

**Analysis of Simulation Trajectories (re-stating the history)**:

1. **Trajectory 1: Kxc8**

- **Initial Move**: Kxc8

- **Opponent's Move**: f8g8

- **Next State**: 2K3k1/3r1p2/5Q2/pp2pN2/P3P3/2P4P/1P3q2/8 w - - 1 47

- **Subsequent Move**: Kb8

- **Opponent's Move**: g8f8

- **Next State**: 1K3k2/3r1p2/5Q2/pp2pN2/P3P3/2P4P/1P3q2/8 w - - 3 48

2. **Trajectory 2: Ka6**

- **Initial Move**: Ka6

- **Opponent's Move**: f8g8

- **Next State**: 2r3k1/3r1p2/K4Q2/pp2pN2/P3P3/2P4P/1P3q2/8 w - - 5 47

- **Subsequent Move**: Kxa5

- **Opponent's Move**: g8f8

- **Next State**: 2r2k2/3r1p2/5Q2/Kp2pN2/P3P3/2P4P/1P3q2/8 w - - 1 48

**Evaluation**:

1. **Kxc8 Trajectory**: - After Kxc8, the board simplifies, and the white king is now in a more exposed position. The black king is still a threat, and the board is less controlled by the white queen. - The subsequent move Kb8 does not significantly improve the position, and the black king continues to be a threat.

2. **Ka6 Trajectory**: - After Ka6, the white king is more protected, and the board is less open. The black king is still a threat, but the white queen has more control over the center and the board. - The subsequent move Kxa5 does not significantly improve the position, and the black king continues to be a threat. **(Superficial evaluation)**

**Conclusion**:

While both moves have their merits, Ka6 provides better protection for the king and maintains more control over the board. Therefore, the updated understanding is that Ka6 is a better move to explore further.

**Next Move: Ka6** **(Surface-level evaluation, Over-indexed on "evaluation form" rather than chess substance)**

---

**Experience Summary/Bootstrapping after RL**

The simulation history has ....

Based on the simulation history, the position after the move Kxc8 would be a position with a queen on the board, while the position after the move Ka6 would be a position with a queen taken. Therefore, the position after the move Kxc8 is better for the white player.

**(Outcome-oriented, Evaluates the resulting material outcome)**

Table 14: Reasoning trace example THREE of SKL-RL before and after end-to-end RL training.

**Game State**: r2r2k1/p4ppp/1p1R3b/n3p3/N7/2P2N2/P3K1PP/3R4 w - - 4 22

**State Understanding Before RL**

The current position is as follows: r2r2k1/p4ppp/1p1R3b/n3p3/N7/2P2N2/P3K1PP/3R4 w - - 4 22
**Key Features**:
1. **White to Move**: White is in the process of checkmating Black.
2. **Material**: White has a rook, two knights, and a king. Black has a king and a rook.
3. **Positional Evaluation**:
- White's rook is on the 6th rank, which is a strong position.
- Black's rook is on the 7th rank, which is a weak position.
- White's knights are well-placed to attack Black's king.
- Black's king is in the center, with no immediate escape routes.
**Strategic Considerations**:
- **Checkmate Threat**: White has a strong checkmate threat with the rook and knights.
- **Rook Control**: The rook on the 6th rank is a key piece, controlling the 6th and 7th ranks.
- **Knight Maneuvering**: The knights can be used to deliver checkmate or put pressure on Black's king.
**Candidate Moves (spamming)**:
1. **Rd8**: Checkmate with the rook.
2. **Rd7**: Checkmate with the rook.
3. **Rxh6**: Captures a piece and puts pressure on Black's king.
4. **Rg6**: Check with the rook, putting pressure on Black's king.
5. **Rf6**: Check with the rook, putting pressure on Black's king.
6. **Rc6**: Check with the rook, putting pressure on Black's king.
7. **Rxb6**: Captures a piece and puts pressure on Black's king.
8. **Nxb6**: Captures a piece and puts pressure on Black's king.
9. **Nc5**: Maneuvering the knight to a key square.
10. **Ng5**: Maneuvering the knight to a key square.
**Reasoning**:
- **Rd8**: This move is the most direct and immediate checkmate threat. If Black's king cannot escape, this will result in checkmate.
- **Rd7**: Similar to Rd8, but slightly less direct. It still puts pressure on Black's king and can lead to a quick checkmate.
- **Rxh6**: Captures a piece and puts pressure on Black's king. This move is less direct but still a strong threat.
- **Nxb6**: Captures a piece and puts pressure on Black's king. This move is also a strong threat but less direct than the rook moves.
**Conclusion**:
The best move is **Rd8**, as it directly leads to checkmate if Black's king cannot escape. This move is the most immediate and decisive.
**Next Move: Rd8 (Prefer to propose pure assertion, not conclusion, spamming candidate moves)**

**State Understanding After RL**

This position is complex and dynamic, with White needing to take advantage of their material superiority and central control.
**Candidate moves**:
- 'Kd3': This is too passive and doesn't address the immediate threats.
- 'Rxd8': This sacrifices the rook but takes out a key black rook and opens up the board.
- 'Kf2': This centralizes the king and prepares to control key central squares.
After careful consideration, sacrificing the rook seems more aggressive and decisive.
'Rxd8' is the most dynamic move. It sacrifices the rook but takes out a key black rook and puts pressure on the black rook's central position.
Therefore, I'll go with:
Rxd8 **(More focused, Material awareness is actively considered)**, **(but miss the checkmate advantage)**

## K.6 Reasoning trace for SKL-RL with self-distillation loss

Reasoning trace of SKL-RL after adding self-distillation objective function $\mathcal{L}_{\text{Distill}}$.

**Game State**:
r1b1k2r/5ppp/2p5/p1q1p1B1/Pp2P1n1/1P3NN1/1P3PPP/3Q1RK1 w kq - 0 17

**[State Understanding]**

Based on the simulation history, our understanding of the game state is as follows: **(imprint from forward KL self-distillation data)**

1. Action **Qd8** appears to be the best action among the three. It resulted in a perfect game termination with a reward of 1.0. This action clearly outperformed the other two actions in terms of long-term value. The simulation history for **Qd8** is overwhelmingly positive, and it is the only action that led to a game termination with a perfect reward. Therefore, we can confidently conclude that Qd8 is the best action to take in this game state. We should not consider this action in future game play as it has demonstrated superior performance and potential for future success. **(reasoning with the distilled experience)**

2. Action **Qd7** did not perform well in the simulation history. Although it did not result in a game termination, it did not yield any positive reward. The simulation history for **Qd7** did not provide any evidence of future value. In fact, the game terminated with a reward of 0.0, indicating that the action did not lead to a successful outcome. The simulation history for **Qd7** did not provide any evidence of future value, and it did not perform as well as **Qd8** in terms of long-term value. Therefore, we can conclude that **Qd7** is not a good action to take in this game state.

3. Action **Qd6** ...[Same inference as 2]. However, we should not completely rule out the possibility that Qd6 could perform better in future game play. We should continue to consider Qd6 as a potential action in future game play, but we should not give it the same level of consideration as Qd8. We should be cautious in our use of Qd6 and only consider it as a potential action when Qd8 is not available. We should also be aware that Qd6 did not perform as well as Qd7 in terms of long-term value. Therefore, we should give Qd6 the lowest level of consideration among the three actions. **(careful and reflective about its own decisions)**

**[In Brief]: reasoning with previous knowledge, but not over-confident.**

---

**[Bootstrapping / Self-Evaluation]**

Based on the simulation history, my updated understanding of the game state is as follows:

### Detailed Analysis **(Recognize multiple subsequent steps in simulation)**:
1. **Move Qd8**:
- Result: Terminated the game with a reward of 1.0.
- **Conclusion**: Qd8 is the best move and the only move that leads to a win. It is the optimal move in this state.
2. **Move Qd7**:
- Opponent's move e8f8 led to a state where the game did not terminate.
- Subsequent moves Qxg4, Qd2, and Rd1 all led to states where the game did not terminate. **(More explicit trajectory analysis)**
- **Conclusion**: Qd7 is not a winning move, and it does not guarantee a win. It is a suboptimal move.
3. **Move Qd6**:
- Opponent's move h8g8 led to a state where the game did not terminate.
- Subsequent moves Qd2 and Ra1 both led to states where the game did not terminate.
- **Conclusion**: Qd6 is not a winning move, and it does not guarantee a win. It is a suboptimal move.
4. **Move Qd5**:
- Opponent's move h8g8 led to a state where the game did not terminate.
- Subsequent moves Qd6 led to states where the game did not terminate.
- **Conclusion**: Qd5 is not a winning move, and it does not guarantee a win. It is a suboptimal move.

### Summary: **(Well-rounded consideration)**
- **Qd8** is the only move that guarantees a win and terminates the game with a reward of 1.0.
- **Qd7, Qd6, and Qd5** do not guarantee a win and do not terminate the game. They all lead to states where the game continues and does not terminate.

Therefore, the updated understanding is:
$\boxed{Qd8}$ is the best move to play in this state. Qd7, Qd6, and Qd5 are suboptimal moves that do not guarantee a win. Qd8 is the only move that solves the game.

# L Error Analysis of Knowledge Bootstrapping

Self-generated state knowledge can be inaccurate, raising two related but different questions. First, can an isolated incorrect assessment propagate and amplify through repeated Knowledge Bootstrapping updates? Second, how sensitive is the scaled SKL-SD training procedure to persistently corrupted successor knowledge? We study the first question on FrozenLake and the second on WebShop.

## L.1 FrozenLake: Error Propagation across Bootstrapping Rounds

To complement the motivating FrozenLake analysis in Section 2, we study whether an incorrect state assessment recursively propagates across repeated Knowledge Bootstrapping rounds. Following the analogy to perturbing a local Bellman target, we use predecessor distance as the backup depth and repeated backward bootstrapping sweeps as the update iterations.

**Error regimes.** We consider two types of corruption:

- **One-shot corruption.** A false successor assessment is inserted before the first backward sweep, after which all subsequent updates use clean assessments. This tests whether an isolated error autonomously propagates or accumulates.
- **Persistent corruption.** The same false assessment is re-inserted before every backward sweep. This tests robustness to a recurring erroneous target.

**Experimental setup.** We use successful trajectories from five $5 \times 5$ slippery FrozenLake maps. For each eligible map, we select a state two transitions before the observed terminal success and inject the false assessment that the state is irrecoverable, that no action can reach the goal, and that its predicted success probability is zero.

Each corrupted branch is paired with a clean branch using the same map, source trajectory, backbone model, prompt, initial memory, generation budget, and environment-evaluation seeds. We perform one-step TD-style backward updates for three rounds and conduct three paired policy evaluations per map and round. For trajectory-level Reflexion, we insert the same false state-anchored statement into the global trajectory memory and evaluate it using the same maps and trial seeds. We use *Qwen3-235B-A22B-Inst* as the backbone model.

**Evaluation.** Unlike a tabular value function, language-valued knowledge does not provide a single scalar prediction error that can be checked against a known $V^\pi$. We therefore require each generated state assessment to expose three structured fields: `recoverable`, `preferred_action`, and `predicted_success_probability`. We track their changes relative to the paired clean branch as internal diagnostics. Our primary correctness measure is the success of the resulting policy in the environment.

For map $m$ and update round $r$, success loss is defined as

$$\Delta_{m,r} = \mathrm{Success}^{\mathrm{clean}}_{m,r} - \mathrm{Success}^{\mathrm{corrupted}}_{m,r}, \tag{12}$$

where a positive value indicates degradation caused by corruption. The cumulative loss is the sum over the three update rounds:

$$\Delta^{\mathrm{cum}}_{m} = \sum_{r=1}^{3} \Delta_{m,r}. \tag{13}$$

We compute $95\%$ confidence intervals using 2,000 paired map-panel bootstrap resamples over the five maps. All rounds and both methods associated with the same map are retained together within each resample.

| Regime | Stateful loss by round | Reflexion loss by round | Stateful cumulative loss [95% CI] | Reflexion cumulative loss [95% CI] | Reflexion−stateful [95% CI] |
|---|---|---|---|---|---|
| One-shot | $-0.067 \to 0.000 \to 0.000$ | $0.000 \to 0.067 \to 0.000$ | $-0.067$ [$-0.200$, $0.000$] | $0.067$ [$-0.533$, $0.600$] | $0.133$ [$-0.333$, $0.600$] |
| Persistent | $0.000 \to 0.000 \to 0.133$ | $0.000 \to 0.133 \to 0.200$ | $0.133$ [$0.000$, $0.267$] | $0.333$ [$-0.600$, $1.600$] | $0.200$ [$-0.733$, $1.533$] |

Table 15: Error propagation under one-shot and persistent corruption on FrozenLake. Success loss is defined as clean minus corrupted policy success, so positive values indicate degradation. Round-wise entries correspond to three successive backward-update rounds. Cumulative entries report the mean and paired-bootstrap $95\%$ confidence interval over five maps.

**Results.** Under one-shot corruption, both methods have zero success loss in the final round. Thus, within this experiment, we find no evidence that a single false assessment autonomously amplifies into increasing behavioral degradation after subsequent updates return to clean knowledge. The negative cumulative point estimate for stateful bootstrapping reflects finite-sample variation and should not be interpreted as a benefit from corruption.

Persistent corruption produces a different pattern. The round-wise success loss increases to $0.133$ for stateful Knowledge Bootstrapping and $0.200$ for trajectory-level Reflexion in the final round. This indicates that repeatedly introduced erroneous targets can produce increasing behavioral degradation.

Stateful bootstrapping has a smaller cumulative-loss point estimate than trajectory-level Reflexion. However, the between-method confidence interval is wide and includes zero. We therefore do not claim a statistically reliable stability difference between the two methods. These results remain preliminary because the experiment contains only five maps, one type of injected error, and three bootstrapping rounds.

### L.2 SKL-SD: Sensitivity to Corrupted Successor Knowledge

We next examine whether the content of successor-state knowledge contributes to the scaled SKL-SD training procedure described in Section 4.1. During WebShop training, we corrupt $50\%$ of the successor assessments supplied to the Knowledge Bootstrapping updates and compare this setting with the standard SKL-SD implementation.

| Method | Step 100 | Step 200 | Step 400 |
|---|---|---|---|
| SKL-SD with $50\%$ corrupted successor knowledge | 0.2675 | 0.2910 | 0.3631 |
| SKL-SD with correct successor knowledge | 0.3375 | 0.3820 | 0.4514 |

Table 16: WebShop performance of SKL-SD with correct and corrupted successor knowledge. In the corrupted setting, $50\%$ of successor assessments used during Knowledge Bootstrapping are replaced with incorrect assessments.

As shown in Table 16, the corrupted setting performs worse at every reported checkpoint. At steps 100, 200, and 400, it obtains scores of $0.2675$, $0.2910$, and $0.3631$, compared with $0.3375$, $0.3820$, and $0.4514$ for standard SKL-SD. Nevertheless, the corrupted variant continues to improve throughout training, indicating that corruption degrades but does not completely prevent learning.

This experiment suggests that the semantic content of successor knowledge contributes to the SKL-SD update rather than serving only as additional prompt context. Incorrect successor assessments weaken the temporal consistency targeted by Knowledge Bootstrapping and reduce downstream task performance.

## M SKL-SD State Coverage and Compute Analysis

State-level Knowledge Bootstrapping introduces additional computation because the agent may update multiple intermediate states within each trajectory. We therefore study how the fraction of bootstrapped states affects both performance and computation. We define *state coverage* as the fraction of states selected for Knowledge Bootstrapping within a trajectory.

This is an *experience-matched compute–performance analysis*, rather than a token-matched comparison: the methods use approximately the same number of experience instances, while the completion-token usage and wall-clock time are allowed to vary. The sweep uses *Qwen2.5-1.5B-Instruct* with state-aggregation budget $N = 3$ and look-ahead horizon $H = 3$. Performance is evaluated on WebShop after 400 training steps.

| Method | WebShop | Avg. bootstrap tokens | Avg. rollout tokens | Cumulative tokens | Wall time | Experience instances |
|---|---|---|---|---|---|---|
| SKL-SD, all states | 0.4426 | 1245 | 971 (585+386) | $1.9 \times 10^6$ | $\sim 20$ h | $\approx 2.0 \times 10^5$ |
| SKL-SD, $1/3$ states | **0.4514** | 430 | 575 (317+258) | $7.0 \times 10^5$ | $\sim 8.5$ h | $2.0 \times 10^5$ |
| SKL-SD, $1/5$ states | 0.4280 | 357 | 420 (221+199) | $5.8 \times 10^5$ | $\sim 7.2$ h | $1.9 \times 10^5$ |
| SKL-SD, root-only MC update | 0.4040 | 180 | 346 (193+153) | $2.6 \times 10^5$ | $\sim 5.5$ h | $2.0 \times 10^5$ |
| $R^3L$ | 0.4120 | 269 | 409 (206+203) | $2.0 \times 10^5$ | $\sim 5$ h | $2.0 \times 10^5$ |
| Reflect-GRPO | 0.3440 | 103 | 377 (276+101) | $1.0 \times 10^5$ | $\sim 3$ h | $2.1 \times 10^5$ |

Table 17: Experience-matched state-coverage and compute analysis on WebShop. State coverage denotes the fraction of states updated during Knowledge Bootstrapping. Rollout tokens are divided into the first rollout and retry in parentheses. Cumulative completion tokens and wall-clock time are measured over training steps 0–400. The experience-instance counts are approximately matched, but token usage and computation are not.

**Effect of state coverage.** The results reveal a non-monotonic relationship between state coverage and performance. Bootstrapping every state is neither necessary nor optimal: the all-state variant obtains a lower score than the one-third-state variant ($0.4426$ versus $0.4514$), while using approximately $2.7\times$ as many cumulative completion tokens and more than twice the wall-clock time. Updating every encountered state can therefore introduce redundant or low-value assessments that add both noise and computation.

At the other extreme, reducing Knowledge Bootstrapping to only the root state is also insufficient. The root-only Monte Carlo update reaches $0.4040$, below both the one-third-state variant and the one-fifth-state variant. The one-fifth-state configuration is cheaper than the one-third variant but also performs worse ($0.4280$ versus $0.4514$). These results suggest that state coverage provides a configurable trade-off: too little bootstrapping limits the propagation of intermediate knowledge, whereas bootstrapping every state can introduce unnecessary cost and distracting information.

**Comparison under matched experience.** Under approximately matched experience-instance counts, the one-third-state SKL-SD variant performs better than the root-only update, $R^3L$, and Reflect-GRPO. However, it also spends substantially more completion tokens and wall-clock time. The comparison should therefore not be interpreted as showing that SKL-SD is more compute-efficient. Instead, it shows that Knowledge Bootstrapping provides an explicit mechanism for allocating additional reflective computation to intermediate state assessments while holding the amount of collected experience approximately fixed.

In this interpretation, state aggregation controls the breadth of empirical evidence used to update a state, while Knowledge Bootstrapping encourages temporal consistency between current and successor-state assessments. State coverage determines how broadly this additional reflective computation is applied across a trajectory.

**Token-aligned comparison.** For completeness, Table 18 compares $R^3L$ and the one-third-state SKL-SD variant at matched cumulative completion-token checkpoints.

| **Method** | $5 \times 10^4$ | $1 \times 10^5$ | $1.5 \times 10^5$ | $2 \times 10^5$ |
|---|---|---|---|---|
| $R^3L$ | 0.321 | 0.382 | 0.406 | 0.412 |
| SKL-SD, $1/3$ states | 0.248 | 0.348 | 0.373 | 0.395 |

Table 18: WebShop performance at matched cumulative completion-token checkpoints. $R^3L$ performs better at every reported checkpoint.

$R^3L$ performs better at every reported token checkpoint. Consequently, the strength of SKL-SD in the experience-matched comparison should not be described as token efficiency. The additional performance observed after 400 training steps is obtained by spending more internal reasoning computation on state aggregation and Knowledge Bootstrapping.

Overall, this analysis supports a narrower interpretation of SKL-SD: it provides a configurable way to scale and organize self-reflection under an approximately fixed experience budget. State coverage exposes an explicit compute-allocation axis, but additional computation is not monotonically beneficial, and its value depends on where Knowledge Bootstrapping is applied.