# SCALING AGENT LEARNING VIA EXPERIENCE SYNTHESIS

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

## ABSTRACT

While reinforcement learning (RL) can empower large language model (LLM) agents by enabling self-improvement through interaction, its practical adoption remains challenging due to costly rollouts, limited task diversity, unreliable reward signals, and infrastructure complexity, all of which obstruct the collection of scalable experience data. To address these challenges, we introduce DREAMGYM, the first unified framework designed to synthesize diverse experiences with scalability in mind to enable effective online RL training for autonomous agents. Rather than relying on expensive real-environment rollouts, DREAMGYM distills environment dynamics into a reasoning-based experience model that derives consistent state transitions and feedback signals through step-by-step reasoning, enabling scalable agent rollout collection for RL. To improve the stability and quality of transitions, DREAMGYM leverages an experience replay buffer initialized with offline real-world data and continuously enriched with fresh interactions to actively support agent training. To improve knowledge acquisition, DREAMGYM adaptively generates new tasks that challenge the current agent policy, enabling more effective online curriculum learning. Experiments across diverse environments and agent backbones demonstrate that DREAMGYM substantially improves RL training, both in fully synthetic settings and in sim-to-real transfer scenarios. On non-RL-ready tasks like WebArena, DREAMGYM outperforms all baselines by over $150\%$. And in RL-ready but costly settings, it matches GRPO and PPO performance using only synthetic interactions. When transferring a policy trained purely on synthetic experiences to real-environment RL, DREAMGYM achieves an additional 64.5% performance gain while using no more than 10% of real-world interactions.

## 1 INTRODUCTION

Autonomous agents based on large language models (LLMs) are being widely adopted across a broad range of tasks given their comprehensive pre-trained semantic knowledge. These agents have already shown promise in applications such as web navigation (Zhou et al.), embodied control (Shridhar et al.), and multi-turn tool use (Yao et al., 2024). However, while these agents can leverage strong language priors to reason and plan, their performance in downstream interactive settings remains limited (Wang et al., 2024). As we step into the *era of experience* (Silver & Sutton, 2025), a promising direction for building more robust and adaptive language agents is reinforcement learning (RL), where agents improve by interacting with environments and bootstrapping from their own experiences (Schulman et al., 2017), as illustrated in figure 1 (a).

Despite its potential, training LLM agents via RL remains highly challenging in practice. The most fundamental barrier is the high cost and low sample efficiency of collecting large-scale, diverse, and informative online interaction data (Wei et al., 2025; Jiang et al., 2025). Real environments often involve long interaction sequences, high computational cost per step, and sparse reward feedback, making it prohibitively expensive to gather sufficient amount of data for modern RL algorithms (Patil et al.; Shao et al., 2024). Beyond computational cost, there is also a lack of diverse, scalable tasks, where most existing environments provide only a limited, static set of instructions, while RL training requires a broad range of tasks for effective exploration (Eysenbach et al., 2018). However, scaling task instructions is inherently difficult, as validating their feasibility often demands costly human expertise (Xue et al., 2025), leaving current environments insufficient for goal-conditioned RL.

The third major barrier is the instability of reward signals. Many interactive settings, such as web pages and GUIs, are highly dynamic and lack consistent behaviors, resulting in noisy, sparse, or even false feedback that hinders stable learning (Deng et al., 2023). Safety concerns further compound these challenges, as certain actions are irreversible (e.g., deleting an item on a real website), and most environments lack reliable reset mechanisms (Zhou et al.). Finally, the infrastructure difficulty of constructing RL-ready environments also remain challenging. Existing systems are heterogeneous and often rely on heavyweight backends like Docker (Jimenez et al.) or virtual machines (Xie et al., 2024b), making large-batch rollout sampling engineering-intensive and costly. These limitations make building general-purpose and scalable systems for training agents with RL an open and pressing challenge.

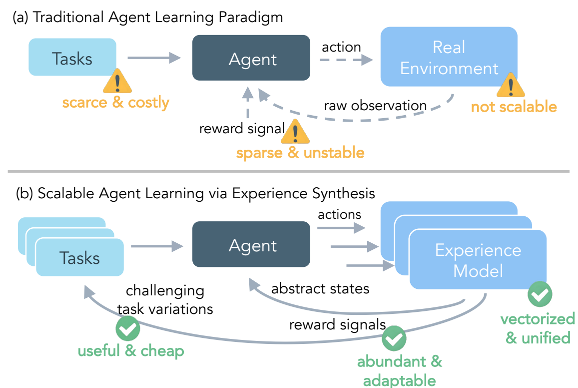


Figure 1: Compared to the traditional agent learning paradigm, DREAMGYM provides the first scalable and effective RL framework with unified infrastructure.

To address these challenges, we propose **DREAMGYM**, a unified and scalable RL framework that synthesizes diverse experience data in an online manner to enable efficient and effective training of LLM agents. At the core of DREAMGYM lies a scalable *reasoning-based experience model* that abstracts environment dynamics into a discrete textual space. By interacting with the agent over multiple turns, it produces consistent transitions and feedback that reflect the consequences of the agent's actions through explicit reasoning. Unlike prior approaches that attempt to reproduce external systems (Chen et al., 2025; Assran et al., 2025), the design of the experience model is grounded in a key insight that *agent training does not require perfectly realistic environments, but rather interaction data that is sufficiently diverse, informative, and causally grounded to acquire knowledge* for the target task. Therefore, powered by strong reasoning, the experience model overcomes the key limitations outlined above and delivers useful experience data for RL training.

To ensure that synthetic experiences are diverse and informative, DREAMGYM equips the experience model with an *experience replay buffer*, from which it retrieves similar yet diverse trajectories to guide its current state prediction. This buffer is seeded with offline knowledge for essential context and is continuously enriched with trajectories generated on-the-fly, co-evolving the experience model with the agent to ensure the produced rollouts aligned with the agent's updated policy for stable training. In parallel, the experience model serves as a *task generator*, identifying valuable tasks with high reward entropy and producing progressively more challenging variations. This design yields an effective curriculum, where agents are consistently exposed to harder problems as their capability improves. By unifying interaction, memory, and adaptive online task generation, DREAMGYM addresses the persistent challenges that have limited RL for LLM agents training: prohibitive cost, scarcity of diverse tasks, unstable reward signals, and heavy infrastructure demands. It reframes training around an environment purpose-built for RL, enabling efficient synthetic training and effective sim-to-real transfer, improving generalization while minimizing reliance on costly real-world interactions.

Comprehensive experiments are conducted to evaluate DREAMGYM across diverse environments and LLM agent backbones. For use cases lacking RL training support (e.g., WebArena (Zhou et al.)), DREAMGYM provides the only viable approach for RL-based agent training, delivering over 150% improvement over all baselines and SOTA methods. In settings where RL is supported but costly, DREAMGYM achieves performance on par with GRPO (Shao et al., 2024) and PPO (Schulman et al., 2017), while training entirely within DREAMGYM without external interactions. Moreover, we introduce DREAMGYM-S2R (sim-to-real), which first trains agents in DREAMGYM using diverse, curriculum-driven synthetic experiences before transferring them to external environments. This approach yields a 64.5% gain compared to training from scratch in real environments while using less than 10% of the external data, providing a scalable warm-start strategy for general-purpose RL.

## 2 Related Work

### 2.1 LLM Agents Reinforcement Learning

RL offers a path to transform LLM agents from static generators into adaptive decision makers. Classical RL algorithms such as policy gradients and actor–critic methods (Williams, 1992; Schulman et al., 2017) have achieved strong results in robotics, games, and control (Silver et al., 2016). More recently, a range of RL paradigms has been proposed to enhance language modeling: RLHF (Bai et al., 2022) and DPO (Rafailov et al., 2023) focus on post-training alignment with human values, while GRPO (Shao et al., 2024) and PPO (Schulman et al., 2017) have been applied for verifiable RL training to improve performance on math and coding tasks. This line of work has further inspired follow-up algorithms such as DAPO (Yu et al., 2025) and Dr. GRPO (Liu et al., 2025), which aim to improve the efficiency and stability of GRPO-style post-training for LLMs.

However, these RL algorithms primarily focus on single-turn language modeling, and extending them to interactive, multi-step agent tasks introduces substantial challenges (Zhang et al., 2025). Long-horizon tasks such as web navigation (Yao et al., 2022; Zhou et al.), operating-system control (Xie et al., 2024b), and multi-tool orchestration (Yao et al., 2024; Xie et al., 2024a) offer limited task sets (Zhou et al., 2025) and sparse rewards (Qi et al., 2024), making exploration inefficient and policy improvement highly unstable. Moreover, real-world dynamics are heterogeneous, noisy, and can easily trigger irreversible unsafe actions (Xue et al., 2025). Although sandboxed environments like WebArena (Zhou et al.) attempt to mitigate these issues with Docker-based isolation, they still lack reliable reset mechanisms and cannot support scalable, parallel data collection for RL. While certain web-specific RL frameworks such as WebAgent-R1 (Wei et al., 2025) and WebRL (Qi et al., 2024) aim to address these limitations through extensive engineering, they remain constrained by the inherent challenges of the underlying real-world environments and thus exhibit low sample efficiency and frequent training collapse. Therefore, a scalable, unified RL framework capable of transferring across real environments without overwhelming engineering effort is urgently needed.

### 2.2 Training Agents with Synthetic Data

Synthetic data has long been used to address the scarcity of expert demonstrations (Zhang et al., 2025). Early approaches relied on scripting oracle trajectories or generating them from stronger teacher models, training agents primarily through imitation learning (Yao et al., 2022; Pahuja et al., 2025; Deng et al., 2023). While effective for distillation, these static trajectories require substantial human labeling and lack diversity and adaptivity. To reduce manual effort, subsequent work (Xu et al.; Ou et al., 2024) leveraged indirect sources of expertise (e.g., online tutorials) to guide trajectory synthesis, while another line of work, including AgentSynth (Xie et al., 2025) and SCA (Zhou et al., 2025), focuses on synthesizing diverse tasks to expand the space of RL exploration. However, these methods still depend on data collection in real environments, which inherently suffer from the scalability issue and are subject to the limitations outlined earlier.

Later research shifted toward building synthetic environments, such as AlphaGo (Silver et al., 2016) and Dreamer (Hafner et al., 2020; Ha & Schmidhuber, 2018), which interact with agents to generate unlimited on-policy experiences. More recently, WebDreamer (Gu et al., 2024) and WebEvolver (Fang et al., 2025) constructed world models to produce environment feedback for agent planning and training. Closest to our setting, UI-Simulator (Wang et al., 2025) also leverages LLMs as a step-wise simulator, but it directly synthesizes raw DOM observations, which is expensive and challenging, and is limited to producing trajectory variations for supervised policy fine-tuning. Similarly, Simia-RL (Li et al., 2025) generates online step-wise interactions using closed-source models with strong reasoning capability (e.g., OpenAI o4-mini), leading to data that are often out-of-domain, extremely costly, and usable only for the current training run. In contrast, DreamGym is significantly cheaper and more scalable: it uses a low-cost open-source model for annotation, trains an experience model once to synthesize transitions in a token-efficient meta-representation space, and then reuses that model to generate unlimited, stable online interaction data for training any agents via RL.

## 3 Preliminaries

### 3.1 Notations

We formalize the agent learning problem as a Markov Decision Process (MDP) (Bellman, 1957), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \gamma, \rho_0)$, where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ denotes the action space. The transition function $T\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ governs the environment dynamics, where $\Delta(\mathcal{S})$ denotes the probability simplex over $\mathcal{S}$. The reward function $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ provides feedback signals for the agent's actions. $\gamma \in [0, 1]$ is the discount factor, and $\rho_0 \in \Delta(\mathcal{S})$ specifies the initial state distribution that includes the task instruction $\tau_0$.

In LLM agent environments, $\tau_0$ is usually a desired task specified by the user in natural language, and states $s \in \mathcal{S}$ encode the environment configuration visible to the agent, such as webpage content, tool outputs, or textual environment descriptions. Actions $a \in \mathcal{A}$ represent discrete operations, including

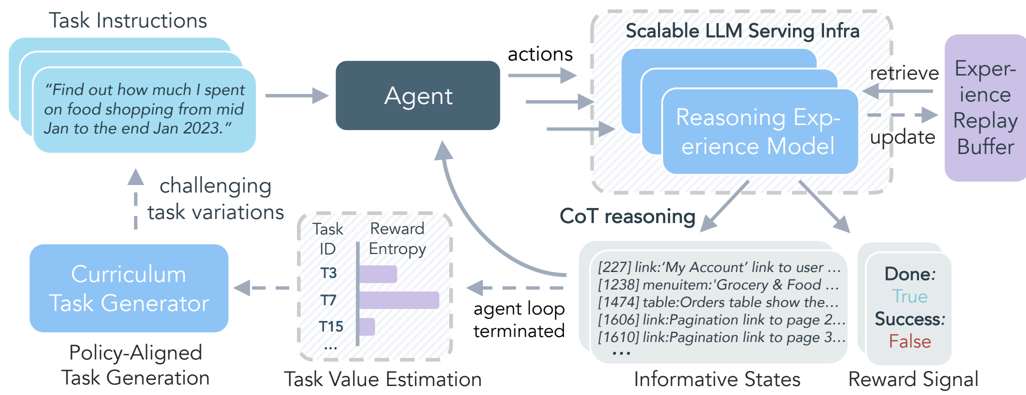


Figure 2: Overview of the proposed DreamGym agent training framework. Given a set of seed tasks, a reasoning-based experience model interacts with the agent to generate informative, diverse tasks and trajectories for RL training. At each step, the agent takes actions based on its current state and receives next states and reward signals derived by the experience model through CoT reasoning based on both interaction history and top-$k$ similar experiences from an active replay buffer. To expose the agent to increasingly informative scenarios, tasks with high reward entropy are proposed by the curriculum task generator for future training. With this unified design, DreamGym addresses both task and reward sparsity while enabling scalable RL with diverse and curriculum-driven environments.

clicking UI elements, invoking external tools, or generating textual responses. The agent maintains a policy $\pi_\theta : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, parameterized by $\theta$, which maps states to distributions over actions.

### 3.2 Agent Learning from Experience

Given a set of online experiences where each experience $\epsilon = \{\tau_0 \mid s_0, a_0, \ldots\}$ consists of a task $\tau_0$ and state-action rollout $\{s_0, a_0, \ldots, s_t, a_t\}$, the goal of RL is to train an agent policy $\pi_\theta$ to maximize the expected cumulative reward, which typically optimized $\theta$ via policy gradient as follows:

$$\nabla J(\theta) = \mathbb{E}_{(s_t, a_t) \sim \pi_\theta} \left[ \nabla \log \pi_\theta(a_t \mid s_t) \cdot \hat{A}(s_t, a_t) \right], \tag{1}$$

where $\hat{A}(s_t, a_t)$ is the *advantage function*, estimating how favorable an action is compared to others.

**Proximal Policy Optimization (PPO).** PPO (Schulman et al., 2017) is a popular policy gradient method that improves stability by computing $\hat{A}$ with *Generalized Advantage Estimation (GAE)*:

$$\hat{A}_t^{\text{PPO}} = \sum_{l=0}^{K-1} (\gamma\lambda)^l \left[ r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l}) \right], \tag{2}$$

where $V(\cdot)$ is a value function approximated by a LLM, and $\lambda$ controls the bias-variance tradeoff.

**Group Relative Policy Optimization (GRPO).** GRPO (Shao et al., 2024) extends PPO by discarding the value function and normalizing advantages within each group of responses $\mathcal{G}$ sampled for the same task instruction. Instead of GAE, the group-relative advantage is defined as:

$$\hat{A}_t^{\text{GRPO}} = (r_t - \text{mean}_{i \in \mathcal{G}}(r_i)) / \text{std}_{i \in \mathcal{G}}(r_i) \tag{3}$$

where $r_t$ is the reward for output $o_t$, $\text{mean}_{i \in \mathcal{G}}(r_i)$ and $\text{std}_{i \in \mathcal{G}}(r_i)$ are mean and standard deviation of rewards from group $\mathcal{G}$. GRPO discards the value function and approximates the advantage using relative normalized rewards, making policy updates more scalable but potentially less sample-efficient. Notably, our proposed DreamGym is orthogonal to specific RL algorithms and focuses on scaling the synthesis of diverse, informative experiences, thereby amplifying the effectiveness of RL training.

## 4 Scaling Agent Learning via Experience Synthesis

To synthesize diverse agent experiences for RL training, DreamGym is built around three key components: (1) a scalable *reasoning experience model* that encodes the meta-dynamics of the target domain to efficiently generate informative trajectories; (2) an *experience replay buffer* that integrates offline environment knowledge with online synthetic transitions, co-evolving with the agent to stay aligned with its updated policy; (3) a *curriculum task generator* that produces progressively challenging variations of high-value tasks selected via a reward-entropy heuristic. We provide a detailed pseudocode illustrating the synthesis and training pipeline in Algorithm 1 in Appendix B.2.

### 4.1 Building Reasoning Experience Models for Agent Learning

For effective RL training, instead of relying on heterogeneous external environments that are costly to interact with and difficult to control, DreamGym adopts a more adaptive and controllable approach by building a LLM-based experience model that can efficiently interact with the agent over multiple turns to generate diverse experiences with consistent outcomes and rich feedback signals for learning. Unlike prior data-hungry and costly approaches that build world models to replicate the real world in raw pixel spaces, we design an efficient reasoning experience model, denoted as $\mathcal{M}_{\text{exp}}$, that operates in an abstract, meta-representational textual space $\mathcal{S}$. The key insight is that synthesizing transitions in this abstract state space can reduce irrelevant dimensions and produce trajectories that are more informative and token-efficient than those derived from raw observations. For example, in a web shopping task, instead of processing raw HTML code, the experience model directly synthesizes clean element listings while discarding irrelevant structural artifacts such as headers and tags. This state-space design makes training the experience model highly sample-efficient, requiring only small pubic trajectory datasets in our experiments, while also enhancing the effectiveness of agent learning.

#### 4.1.1 Inference for Experience Rollout Collection

Notably, we find that beyond the current state-action pair, three additional contexts are important for improving state quality: (1) *interaction history* $\{(s_i, a_i)\}_{t=0}^{T}$, which incorporates the past trajectory in the context window to help maintain state consistency across multiple turns; (2) *task instruction* $\tau$, which conditions the experience model on the current goal, enabling it to interpret actions w.r.t. task objectives and thereby predict both state transitions and rewards more accurately; (3) *past experiences*, which are top-$k$ demonstrations $\{d_j\}_{j=1}^{k}$ retrieved from the replay buffer based on semantic similarity with the state-action pair, i.e., $\{d_j\}_{j=1}^{k} = \text{Top}_k(\cos(\phi(s_t, a_t), \phi(s_i, a_i)))$, where $\phi(\cdot)$ denotes an arbitrary semantic encoder. Leveraging knowledge this way reduces hallucinations and improves factuality for knowledge-intensive state predictions. Therefore, given these inputs, the experience model predicts the next state $s_{t+1}$ and reward $r_{t+1}$ via chain-of-thought (CoT) (Wei et al., 2022):

$$(s_{t+1}, r_{t+1}) = \mathcal{M}_{\text{exp}}\Big(R_t \Big| \{(s_i, a_i)\}_{i=0}^{t}, \{d_j\}_{j=1}^{k}, \tau\Big). \tag{4}$$

where $R_t$ is an explicit reasoning trace produced by the experience model that guides the state transition. With such reasoning, it predicts the most consistent and informative transition and feedback that reflects the consequence of the agent action. For example, if the action is invalid, it transitions to a failure state and assigns a zero reward to signal the error, and vice versa. In our experiments, following (Feng et al., 2025), we adopt an outcome-based reward scheme, assigning $r = 1$ only at the final step when the task is successfully completed and $r = 0$ in all other cases.

#### 4.1.2 Training Experience Models to Reason

Benefiting from the abstract state-space design, training the experience model is highly sample-efficient and requires only limited data from the real environment. In practice, abundant offline trajectory datasets from public benchmarks such as the WebArena Leaderboard are sufficient for training. Our experience model distills such offline knowledge and then serves as a bridge to interact with the agent online for RL training.

Concretely, given a trajectory dataset $\mathcal{D} = \{(s_t, a_t, s_{t+1}, r_{t+1})\}$, each transition is annotated with an explicit reasoning trace $R_t^*$ by LLM (prompt shown in Appendix E.1), which explains why the action $a_t$ taken in state $s_t$ consequently leads to the next state $s_{t+1}$ and reward $r_{t+1}$ given the available contexts. To distill this knowledge, we train $\mathcal{M}_{\text{exp}}$ via SFT with a joint objective over reasoning generation and next-state prediction:

$$\mathcal{L}_{\text{SFT}} = \mathbb{E}_{(s_t, a_t, s_{t+1}, R_t^*) \sim \mathcal{D}}\Big[-\log P_\theta(R_t^* \mid s_t, a_t, \mathcal{H}_t, \mathcal{D}_k) - \log P_\theta(s_{t+1} \mid s_t, a_t, R_t^*, \mathcal{H}_t, \mathcal{D}_k)\Big], \tag{5}$$

where $\mathcal{H}_t$ denotes the interaction history, $\mathcal{D}_k$ denotes the retrieved top-$k$ demonstrations, and $\theta$ denotes the parameters of $\mathcal{M}_{\text{exp}}$. This objective ensures that the model (i) learns to generate faithful reasoning traces that explain the causal effect of an action, and (ii) leverages these traces to predict consistent and informative next states. By doing so, the experience model not only imitates expert trajectories but also acquires the ability to generalize reasoning for novel rollouts during RL training.

### 4.2 Curriculum-based Task Generation

Diverse, curriculum-aligned task instructions are important for RL agents to acquire knowledge (Zhou et al., 2025). However, scaling task collections is costly, as it requires significant human effort to verify the feasibility of each task in the target environment. DREAMGYM inherently alleviates this burden by adapting to arbitrary new tasks within the target domain through synthetic multi-turn transitions. Building on this capability, we propose *curriculum-based task generation*, where the same experience model actively generates new tasks as variations of a set of $m$ seed tasks:

$$\tau_t = \mathcal{M}_{\text{task}}(\{\tau^i_{t-1}\}^m_{i=1}), \tag{6}$$

where $\mathcal{M}_{\text{task}}$ shares parameters with $\mathcal{M}_{\text{exp}}$. Specifically, the seed tasks are chosen based on two criteria: (1) they are sufficiently challenging for the current agent policy, thereby maximizing information gain; (2) they are well-defined, such that unrealistic or malformed tasks can be discarded.

To satisfy both conditions, we introduce a *group-based reward entropy* as a criteria for selecting high-quality and challenging tasks. Formally, for a task $\tau$, we define its value

$$\mathcal{V}_\tau = \frac{1}{n}\sum_{i=1}^{n}\left(r^i - \bar{r}\right)^2, \quad \text{where } \bar{r} = \frac{1}{n}\sum_{i=1}^{n} r^i, \tag{7}$$

where $r^i$ are the outcome rewards from $n$ rollouts of task $\tau$ within the group $\mathcal{G}$. For GRPO, $\mathcal{G}$ can simply be the training group, while for PPO, tasks can be first clustered using a semantic embedder, and each cluster essentially forms a group $\mathcal{G}$ from which task variations can be generated. Notably, a non-zero variance in $\mathcal{G}$ indicates that the agent observes both successes and failures on the task, signaling that the task is **feasible yet challenging**. A task reaches maximum entropy when **successes and failures are evenly balanced** in $\mathcal{G}$, providing the greatest information gain for credit assignment. This observation is consistent with recent findings that LLMs learn most effectively from tasks of intermediate difficulty (Gao et al., 2025). Thus, by feeding such high-entropy tasks into $\mathcal{M}_{\text{task}}$, we generate progressively more challenging variations to enhance agent exploration and learning.

To stabilize training, we introduce a hyperparameter $\lambda$ that bounds the proportion of synthetic tasks sampled per iteration. This preserves sufficient coverage of the original task distribution while directing exploration toward the current policy's weakness regions for curriculum-based improvement.

### 4.3 LEARNING FROM SYNTHETIC EXPERIENCES

**Policy training in synthetic environments.** As shown in figure 2, DREAMGYM begins with a seed task set and generates multi-turn rollouts for each task by alternating between the agent policy, which selects actions from states, and the experience model, which predicts next states conditioned on the agent action, history, and task context (as in section 4.1.1). The collected rollouts are used with standard RL algorithms (as in section 3.2) to update the policy. After each iteration, the experience model augments the task set by generating variations of challenging tasks with high reward entropy (as in section 4.2). This cycle of interaction, training, and curriculum expansion continues until convergence or a predefined training budget is reached. Furthermore, we provide an analytical lower bound of the policy improvement in real environments when training with purely synthetic experiences from DREAMGYM under trust-region assumptions, as detailed in Appendix D.1.

**Sim-to-real policy transfer.** We further extend DREAMGYM to a sim-to-real (S2R) setting, where the agent policy is first trained with synthetic experiences and then transferred to RL in real environments. Pretraining in synthetic environments expands exploration coverage across diverse tasks and allows the agent to acquire broad knowledge at low cost, providing a strong initialization that makes subsequent real-environment learning more sample-efficient (Da et al., 2025). To enable seamless transfer, we ensure consistency of the state space between synthetic and real environments by applying the same rule-based mapping function or a lightweight fine-tuned model (Lee et al.).

## 5 EXPERIMENTS

### 5.1 EXPERIMENTAL SETUP

We evaluate DREAMGYM on a diverse suite of agentic benchmarks and LLM backbones of varying sizes and model families to assess its generalizability and effectiveness in supporting RL for generic agent tasks while reducing costly interactions.

**Evaluation environments.** We consider three challenging agent benchmarks that span diverse domains, complexities, and levels of RL readiness: (1) WebShop (Yao et al., 2022), which requires reasoning to refine search queries and accurately identify products to complete e-commerce tasks; (2) ALFWorld (Shridhar et al.), which involves multi-turn tool-based embodied control to navigate 3D environments; (3) WebArena-Lite (Zhou et al.), which offers realistic web interaction interfaces but is not RL-ready, as it inherently lacks scalable data collection and environment reset mechanisms while incurring high computational costs. This mixture of environments allows us to evaluate DREAMGYM both in settings where RL is feasible but expensive, and where RL training is not yet tractable.

Table 1: Comparison of DreamGym with various agent training algorithms. We evaluate four groups: (i) *offline imitation learning algorithms*: SFT, DPO; (ii) *online RL algorithms in real-world environments*: GRPO, PPO; and (iii) DreamGym, where agents are trained via the same RL algorithms but with purely synthetic experiences; (iv) DreamGym-S2R, where agents are first trained with synthetic experiences and then transfer to RL in real environments. Real data indicates the number of individual transitions (a trajectory often has $\sim$10 steps). Best performance is bolded.

| Algorithm | Real Data | WebShop | | | ALFWorld | | | WebArena | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | L3.2-3B | L3.1-8B | Q2.5-7B | L3.2-3B | L3.1-8B | Q2.5-7B | L3.2-3B | L3.1-8B | Q2.5-7B |
| *Offline Imitation Learning* | | | | | | | | | | |
| SFT | 20K | 32.0 | 35.1 | 32.9 | 61.7 | 68.0 | 71.8 | 8.7 | 11.3 | 13.0 |
| DPO | 40K | 35.9 | 31.0 | 34.8 | 63.3 | 63.9 | 61.1 | 8.9 | 10.3 | 11.3 |
| *GRPO* | | | | | | | | | | |
| Traditional | 80K | 62.1 | 65.0 | 66.1 | **65.3** | 70.9 | 79.8 | 12.7 | 10.0 | 16.0 |
| DreamGym | 0 | 59.3 | 63.9 | 68.3 | 62.1 | 66.3 | 71.0 | 35.1 | **43.0** | 45.1 |
| DreamGym-S2R | 5K | **70.5** | **75.0** | **72.1** | 65.0 | **75.9** | **82.4** | **38.0** | 37.1 | **47.3** |
| *PPO* | | | | | | | | | | |
| Traditional | 80K | 59.9 | **64.2** | 68.1 | 47.0 | 72.9 | **81.1** | 13.9 | 15.7 | 19.2 |
| DreamGym | 0 | 60.5 | 58.1 | 65.0 | 40.5 | 70.8 | 72.7 | **32.0** | 45.0 | 37.8 |
| DreamGym-S2R | 5K | **66.0** | 63.9 | **73.7** | **49.1** | **73.3** | 79.9 | 30.3 | **48.3** | **41.0** |

**Agent backbones.** We instantiate agents from different model families and sizes: Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Qwen-2.5-7B-Instruct (Team, 2024).
**Baselines.** We consider two traditional training strategies for agents. (1) Offline imitation learning: supervised fine-tuning (SFT), direct preference optimization (DPO) (Rafailov et al., 2023); (2) Online RL in real environments (traditional): GRPO (Shao et al., 2024), PPO (Schulman et al., 2017).

**Implementation details.** All main results are reported with the experience model trained from Llama-3.1-8B-Instruct (see section 4.1). To demonstrate that DreamGym can be applied to different RL algorithms, we first evaluate both GRPO and PPO entirely within DreamGym, without any real interactions. We further evaluate a hybrid scenario, DreamGym-S2R, where synthetic training is followed by a small-scale RL phase that requires only a limited number of rollouts in the original environments, demonstrating the effectiveness of using DreamGym as a mid-training stage to improve both sample efficiency and the attainable performance after transfer. Detailed parameter settings for each scenario are provided in Appendix A.

### 5.2 Main Results

**Non-RL-ready environment.** DreamGym demonstrates the most significant advantage in environments where large-scale RL infrastructure is not available such as WebArena (Zhou et al.). Unlike existing attempts that fail to make RL effective due to environment limitations, agents trained purely in DreamGym achieve success rates exceeding 30% across all backbones (Table 1), whereas zero-shot RL baselines suffer from limited exploration diversity and sparse reward signals in the original environments. These results demonstrate that DreamGym is not merely a surrogate for costly rollouts, but a mechanism that makes RL training feasible in domains that were previously intractable due to inherent task and engineering constraints.

**RL-ready environments.** On WebShop (Yao et al., 2022) and ALFWorld (Shridhar et al.), DreamGym-trained agents perform on par with GRPO and PPO agents trained on 80K real interactions, despite using only synthetic rollouts. The ability to match strong RL baselines in RL-ready environments without external interactions underscores that DreamGym produces transitions and rewards that are not only coherent and meaningful, but also sufficient for stable policy improvement. Furthermore, when a small-scale RL phase with an affordable number of real rollouts (5k) is applied to the policy mid-trained in the synthetic environment, DreamGym-S2R consistently outperforms both GRPO and PPO baselines trained from scratch in the real environments. This validates the hypothesis that synthetic training can serve as an efficient warm-start strategy that establishes a strong foundation for more sample-efficient RL in the real environment.

**Sample efficiency and training cost.** Training efficiency is further illustrated in figure 3 *Left*, where DreamGym achieves substantial performance gains on WebArena (Zhou et al.) while reducing training effort (including both rollouts sampling time and GPU hours) to roughly one-third or even one-fifth of RL baselines in the real environment. The efficiency gain arises from both the dense feedback offered by curriculum-based rollout synthesis and the lightweight abstract state transitions produced by unified experience models hosted by scalable LLM services, which substantially reduce sampling cost and avoid heterogeneous environment bottlenecks, suggesting that DreamGym is not only a practical solution for RL training in complex and expensive environments, but also as a scalable way to generate low-cost data for stable policy improvement.

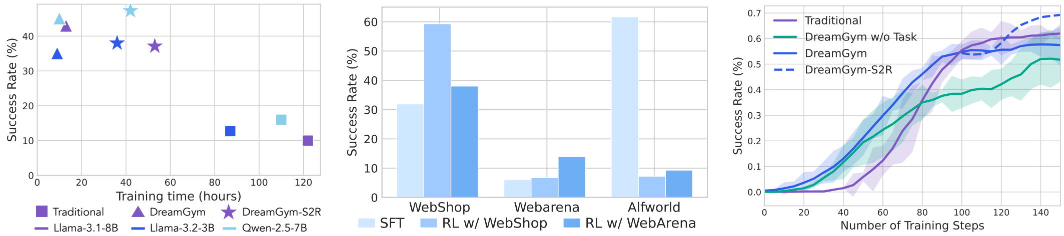


Figure 3: (1) *Left:* Comparing the agent performance (success rate %) on WebArena (Zhou et al.) w.r.t. total training time across different training strategies and backbones. (2) *Middle:* Evaluating the cross-domain transferability of the agent policy trained via DREAMGYM with seed tasks from a different environment. (3) *Right:* Comparing the agent performance on WebShop (Yao et al., 2022) w.r.t. number of training steps across different training strategies.

# 6 PERFORMANCE ANALYSIS AND ABLATION STUDIES

## 6.1 TRAINING CURVE ANALYSIS

figure 3 *Right* compares training curves across Webshop (Yao et al., 2022) under different setups. Specifically, the success rate improves much more rapidly within the first 40 steps, showing that synthesized trajectories offer more informative gradients than sparse real rollouts. This further highlights the role of synthetic experiences in shaping a strong initialization that real rollouts cannot. Another observation is the reduced variance in learning dynamics. Baseline curves exhibit larger oscillations caused by sparse or unstable rewards, while DREAMGYM curves remain smoother across runs, which suggests that synthesized trajectories provide not only denser but also more consistent feedback, mitigating the training instabilities commonly reported in WebShop (Yao et al., 2022) and ALFWorld (Shridhar et al.).

## 6.2 ABLATION ON TASK GENERATOR

Table 2: Average success rates (%) on the different components of DREAMGYM.

| Method | WebShop | WebArena |
|---|---|---|
| DREAMGYM | 63.9 | 43.0 |
| w/o Exp. Replay | 59.2 | 38.1 |
| w/o Exp. Reasoning | 55.8 | 33.9 |
| w/o Task Generation | 57.3 | 31.7 |

The curriculum-based task generator plays an important role in learning progress. As shown in figure 3 *Right*, removing this component causes agents to make some initial progress but then plateau more quickly in the WebShop (Yao et al., 2022) scenario. Similarly, Table 2 shows that removing the task generator leads to a 6.6% and 6.0% drop in success rate compared with the full DREAMGYM configuration in WebShop (Yao et al., 2022) and WebArena (Zhou et al.), respectively.

These findings support our discussion in section 4.2: without adaptive task generation, the replay buffer may saturate with low-entropy, repetitive trajectories, which limits the diversity of experiences and stalls exploration. In contrast, the task generator continually produces progressively challenging, high-value tasks that push the agent beyond its current capability. This ongoing curriculum keeps the replay buffer informative and encourages exploration, ultimately yielding higher final success rates and better sample efficiency.

## 6.3 ABLATION ON EXPERIENCE MODEL

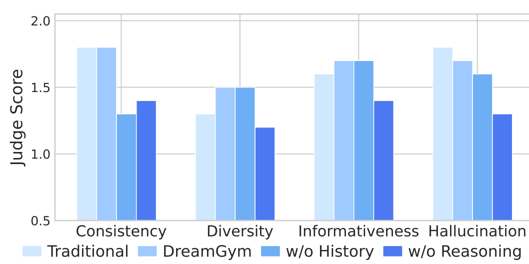


Figure 4: Evaluation of the experience model across key criteria using GPT-4o as the judge. We randomly sample 100 trajectories and prompt the model to assign discrete scores in $\{0, 1, 2\}$ across four criteria, as detailed in Appendix E.4.

figure 4 demonstrates a detailed comparison of experiences generated by four variants of the experience models: a traditional real environment model, DREAMGYM, DREAMGYM without access to past trajectory history (w/o History), and DREAMGYM without reasoning (w/o Reasoning). We evaluate each variant along four criteria: consistency, diversity, informativeness, and hallucination, using GPT-4o (Hurst et al., 2024) as a judge. As detailed in Appendix E.4, the judge assigns discrete scores in $\{0, 1, 2\}$ for each criteria, where higher values indicate better performance. For the first three metrics, larger scores mean more consistent, diverse, and informative; for hallucination, a score of 2 means no hallucination, while 0 indicates more factual errors.

The results highlight the role of each component. Removing trajectory history (w/o History) significantly reduces consistency: without awareness of prior turns, the model often drifts off-topic and breaks causal coherence in multi-step interactions. Removing reasoning (w/o Reasoning) mainly

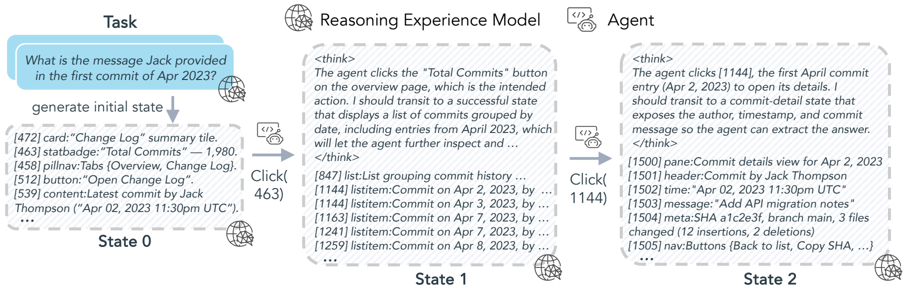


Figure 6: A case study of a trajectory sampled with DREAMGYM in WebArena. Starting from a synthetic instruction, the experience model reasons over the agent's action to produce future states.

hurts informativeness of the states and increases hallucination: without reasoning capabilities, the generated experiences tend to become shallow and less factually grounded. Notably, removing experience reasoning also leads to a substantial drop in overall performance, as shown in Table 2. In contrast, the full DREAMGYM achieves the best or near-best performance across all metrics, confirming that history and reasoning provide complementary benefits. More specifically, history preserves temporal and causal structure, while reasoning enhances depth and factual reliability. This validates that the experience model must operate in a structured, reasoning-driven manner to maintain both diversity and fidelity of trajectories.

### 6.4 Ablation on Experience Model Backbones and Offline Training Data

figure 5 investigates how the success rate of the experience model varies with both the amount of offline training data and the choice of model backbone, evaluated on (a) WebShop and (b) WebArena. We first observe that the experience model is highly data-efficient. Even with a very limited number of offline samples (2k-10k), it already reaches competitive performance. On WebShop, for example, the Llama-3.1-8B exceeds 50% success rate with only 10k samples, indicating that large-scale offline datasets are not strictly necessary for effective experience synthesis. Next, we find that smaller backbones remain viable. Although Llama-3.2-3B underperforms the 8B model, it improves steadily as more data becomes available, reaching about 55% success on WebShop with 20k samples, which suggests that lightweight models can still serve as practical experience generators when computational resources are constrained. Finally, in the extreme low-data regime, pretrained world knowledge becomes particularly valuable. On WebArena, WebDreamer (Gu et al., 2024) (a fine-tuned web world model) achieves around 19% success, initially outperforming the Llama-3.1-8B model at smaller data scales but eventually converging as the number of offline samples increases.

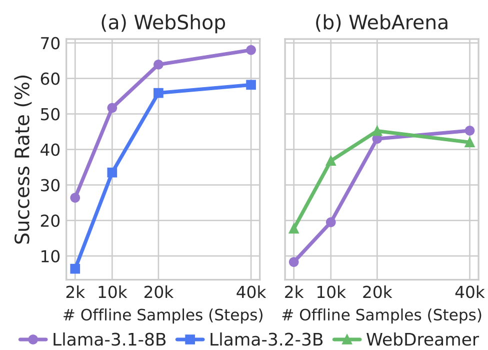


Figure 5: Evaluation of the experience model across different number of offline training data size (transition step) and backbone.

### 6.5 Case Study of Synthetic Experiences from DreamGym

figure 6 illustrates how the reasoning experience model generates a synthetic task and progressively predicts states based on the agent's actions. Specifically, it predicts each state through explicit chain-of-thought reasoning that incorporates the agent's action, task instruction, and interaction history, producing next states that consistently ground the action and accurately reflect its consequences.

Additional experiment results, ablations, and analysis can be found in Appendix C.

## 7 Conclusion

We introduced DREAMGYM, a framework that reduces the high cost of real-environment rollouts in RL for language agents by generating scalable, reasoning-driven synthetic experiences. DREAMGYM compresses environment dynamics into a reasoning-based experience model that produces state transitions and adaptive curricula, creating challenging yet solvable tasks tailored to the agent's evolving policy. Experiments across diverse environments and model backbones show consistent gains in both synthetic and sim-to-real settings, driven by the synergy of reasoning-based modeling, replay-buffer grounding, and curriculum generation. More broadly, our results suggest that the key bottleneck in RL for LLM agents lies in the quality and structure of interaction data. By treating environments as generators of structured, reasoning-rich experiences rather than mere simulators, DREAMGYM enables more scalable, sample-efficient, and generalizable RL for agents.

## Usage of Large Language Models

The language in this paper was at times polished with the assistance of an LLM. The model was not used for research ideation, experimental design, or data analysis.

## References


Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Hyungjoo Chae, Sunghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, et al. Web-shepherd: Advancing prms for reinforcing web agents. *arXiv preprint arXiv:2505.15277*, 2025.

Delong Chen, Theo Moutakanni, Willy Chung, Yejin Bang, Ziwei Ji, Allen Bolourchi, and Pascale Fung. Planning with reasoning using vision language world model. *arXiv preprint arXiv:2509.02722*, 2025.

Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pp. 41–75. Springer, 2018.

Longchao Da, Justin Turnau, Thirulogasankar Pranav Kutralingam, Alvaro Velasquez, Paulo Shakarian, and Hua Wei. A survey of sim-to-real methods in rl: Progress, prospects and challenges with foundation models. *arXiv preprint arXiv:2502.13187*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with coevolving world model. *arXiv preprint arXiv:2504.21024*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Zhaolin Gao, Joongwon Kim, Wen Sun, Thorsten Joachims, Sid Wang, Richard Yuanzhe Pang, and Liang Tan. Prompt curriculum learning for efficient llm post-training. *arXiv preprint arXiv:2510.01135*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. Verltool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Dongjun Lee, Juyong Lee, Kyuyoung Kim, Jihoon Tack, Jinwoo Shin, Yee Whye Teh, and Kimin Lee. Learning to contextualize web pages for enhanced decision making by llm agents. In *The Thirteenth International Conference on Learning Representations*.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*, 2025.

Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Offer Akrabi, Aviad Sela, Asaf Adi, and Nir Mashkif. Towards enterprise-ready computer using generalist agent. *arXiv preprint arXiv:2503.01861*, 2025.

Tianyue Ou, Frank F Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. *Advances in Neural Information Processing Systems*, 37: 91618–91652, 2024.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Awadallah. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of ACL*, 2025.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Junhong Shen, Atishay Jain, Zedian Xiao, Ishan Amlekar, Mouad Hadji, Aaron Podolny, and Ameet Talwalkar. Scribeagent: Towards specialized web agents using production-scale workflow data. *arXiv preprint arXiv:2411.15004*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.

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

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö Arık. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv preprint arXiv:2501.10893*, 2025.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Yiming Wang, Da Yin, Yuedong Cui, Ruichen Zheng, Zhiqian Li, Zongyu Lin, Di Wu, Xueqing Wu, Chenchen Ye, Yu Zhou, et al. Llms as scalable, general-purpose simulators for evolving digital agent training. *arXiv preprint arXiv:2510.14969*, 2025.

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

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: a benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 54590–54613, 2024a.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. *arXiv preprint arXiv:2506.14205*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024b.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations*.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. *arXiv preprint arXiv:2410.13825*, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

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*, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *arXiv preprint arXiv:2506.01716*, 2025.

# A DETAILED EXPERIMENT SETTINGS

In this section, we provide implementation details for each environment.

## A.1 WEBSHOP

WebShop (Yao et al., 2022) is a large-scale agent benchmark designed to study language grounding in interactive environments. It simulates a realistic e-commerce website with 1.18M real-world products and 12,087 crowd-sourced natural language instructions, where agents must search, customize, and purchase items. The environment poses challenges such as interpreting compositional product requirements, reformulating queries, handling noisy webpage text, and strategically exploring diverse page types.

**RL Baseline Setup.** We follow the standard setup and hyperparameter settings from Verl-Agent (Feng et al., 2025) and perform full-parameter fine-tuning for all three agent backbones in our experiments.

**DREAMGYM Settings.** To train the reasoning experience model, we construct a dataset by combining 1,600 human demonstration trajectories from the official WebShop repository with an additional 2,000 trajectories collected using an oracle agent and random exploration. We fix the test set to ensure evaluation stability and collect training trajectories only from the remaining tasks to avoid test-set contamination. Each transition is further augmented with a reasoning trace generated by a strong teacher LLM, resulting in the dataset used for fine-tuning.

**Computation Resources.** All experiments, including both baselines and ours, are conducted on 8 nodes with A100 GPUs and 4 nodes with H100 GPUs.

## A.2 ALFWORLD

ALFWorld (Shridhar et al.) is a text-and-embodied benchmark with hand-crafted task instructions designed for studying language grounding and cross-modal transfer. It pairs abstract text interactions from TextWorld with photo-realistic, physics-based execution in ALFRED/AI2-THOR, spanning six household task families (e.g., Pick & Place, Clean & Place, Heat/Cool & Place) with 3,553 training tasks and seen/unseen splits across 120 rooms. Agents issue high-level textual actions (goto, open, take, clean/heat/cool, put) that must be realized as low-level visuomotor controllers, facing challenges such as partial observability, object search and manipulation, mapping language to action preconditions and affordances, and bridging the gap between abstract plans and physical feasibility.

**RL Baseline Setup.** We adopt the standard setup and hyperparameter settings from Verl-Agent (Feng et al., 2025) and perform full-parameter fine-tuning for all three agent backbones.

**DREAMGYM Settings.** We follow the default `ALFWorld` split (Shridhar et al.) with the `TextWorld` setup (Côté et al., 2018) under the Verl-Agent framework. From the training split, we extract 3,200 expert demonstration trajectories paired with task instructions, and additionally sample 2,000 offline trajectories using both oracle and random policies. These datasets form the basis for training the reasoning experience model. Each transition is further augmented with a reasoning trace generated by a powerful LLM, which is used for fine-tuning.

**Computation Resources.** All experiments, including both baselines and ours, are conducted on 8 nodes with A100 GPUs and 4 nodes with H100 GPUs.

## A.3 WEBARENA

WebArena (Zhou et al.) is a self-hosted, realistic web environment for training and evaluating autonomous agents across fully functional sites such as e-commerce, social forums, collaborative software development (GitLab), and content management, which are augmented with tools (map, calculator, scratchpad) and knowledge bases (e.g., offline Wikipedia, manuals). It provides 812 long-horizon tasks expressed as high-level natural language intents and evaluates agents by functional correctness rather than matching action traces, supporting multi-tab browsing and a rich action space (click, type, navigate, tab operations).

**Training Set Split.** Since the full evaluation set in `WebArena` is large and contains many similar tasks, we follow prior work (Qi et al., 2024; Wei et al., 2025) and evaluate agents on `WebArena-Lite` (Liu et al., 2024), a more balanced subset of 165 high-quality, challenging tasks selected from the original 812. The remaining 647 tasks, excluding those in the evaluation set, are used for training.

**RL Baseline Setup.** Since no reliable open-source RL infrastructure exists for WebArena (Qi et al., 2024), we follow the standard workflow in Verl-Agent (Feng et al., 2025) and implement a vanilla RL pipeline for WebArena. The baseline collects action trajectories via `browsergym` (Zhou et al.) while hosting the WebArena websites on AWS servers, supporting both PPO and GRPO. Despite extensive engineering effort, we are able to operate only four AWS servers, enabling at most four parallel interaction sessions, which in turn imposes a significant bottleneck on training throughput.

During RL sampling, we sequentially sweep through the task set and manually restart the servers and reset the environments once all tasks have been visited once, in order to avoid cross-task interference. Nevertheless, we observe that some trajectories fail to execute properly and that certain tasks are incorrectly judged by WebArena's original evaluation function, a known issue also reported in prior work (Qi et al., 2024; Chae et al., 2025). For fairness, we retain all collected trajectories and use them as-is for RL training.

**DREAMGYM Settings.** To obtain offline trajectories for training the reasoning experience model, we extract successful demonstrations from the highest-performing agents on the public WebArena leaderboard. Specifically, we use agents that incorporate accessibility tree information in their observations, including IBM CUGA (Marreed et al., 2025), ScribeAgent (Shen et al., 2024), Learn-by-Interact (Su et al., 2025), and AgentOccam (Yang et al., 2024). To mitigate distribution imbalance, we additionally collect trajectories generated by a high-performing agent and by a random policy. In total, we obtain 4,800 offline trajectories for training. Each transition is then augmented with a reasoning trace generated by a powerful LLM, forming the dataset for fine-tuning the reasoning experience model.

**Computation Resources.** All experiments, including both baselines and ours, are conducted on 8 nodes with A100 GPUs and 4 nodes with H100 GPUs.

# B ADDITIONAL DETAILS OF DREAMGYM

## B.1 HYPERPARAMETER CONFIGURATIONS

The hyperparameters for training and serving the experience model are reported in Table 3. The hyperparameters for training agents via RL are reported Table 4.

Table 3: Hyperparameter settings for experience model.

| Parameters | Value |
|---|---|
| Annotation model | Llama-3.3-70B-Instruct |
| Random seed | 42 |
| Top-$k$ examples | 2 |
| Synthetic Task ratio $\lambda$ | 0.4 |
| Temperature | 0.2 |
| Outcome-based reward | True |

Table 4: Hyperparameter of training agents via RL.

| Parameters | Value |
|---|---|
| Random seed | 42 |
| Mini batch size | 64 |
| Micro batch size | 8 |
| Training batch size for WebShop | 16 |
| Training batch size for AlfWorld | 16 |
| Training batch size for WebArena | 8 |
| KL loss coef | 0.01 |
| Learning rate | $1e-6$ |

## B.2 TRAINING PIPELINE OF DREAMGYM

We provide detailed pseudocode in Algorithm 1 to formalize the data synthesis procedure and RL training pipeline in DREAMGYM, covering all critical steps, including: (i) initialization of all components, including the agent policy (initialized from a base model), the replay buffer (seeded with offline trajectories), and the task pool (initialized with a set of seed tasks); (ii) synthesizing multi-turn rollouts by iteratively querying the reasoning-based experience model using the current state–action pair, task instruction, interaction history, and top-$k$ retrieved experiences; (iii) updating the agent policy with an RL optimizer using the collected synthetic experience data; and (iv) computing group-based reward entropy to drive curriculum task generation and updating both the task pool and replay buffer throughout training.

**Algorithm 1** RL Data Synthesis and Training Pipeline of DREAMGYM

**Require:** Experience model $\mathcal{M}_{\text{exp}}$, agent policy $\pi_\theta$, replay buffer $\mathcal{B}$, active task pool $\mathcal{T}$, seed task set $\mathcal{T}_{\text{seed}}$, max episode length $H$, synthetic-task ratio $\lambda$, RL optimizer (e.g., GRPO, PPO)

Initialize policy $\pi_\theta$ ▷ Initialize agent policy from a base LLM
Initialize replay buffer $\mathcal{B} \leftarrow \mathcal{D}_{\text{off}}$ ▷ Offline trajectories from public benchmarks
$\mathcal{T} \leftarrow \mathcal{T}_{\text{seed}}$ ▷ Initialize task pool with seed task instructions
**while** not converged or exceeded maximum epochs **do**
  $\mathcal{R} \leftarrow \emptyset$ ▷ Batch of synthetic rollouts
  **for** each training group $\mathcal{G}$ **do** ▷ GRPO group or PPO task cluster
    Sample tasks $\{\tau_i\}_{i=1}^m \sim \mathcal{T}$ ▷ Mix real and synthetic tasks by ratio $\lambda$
    **for** each task $\tau$ in $\{\tau_i\}$ **do**
      $s_0 \leftarrow \text{INITSTATE}(\tau)$ ▷ Initial state in abstract space $\mathcal{S}$
      $\zeta_\tau \leftarrow \emptyset$
      **for** $t = 0$ to $H-1$ **do** ▷ Experience synthesis loop
        $a_t \sim \pi_\theta(\cdot \mid s_t, \tau)$ ▷ Agent action conditioned on task
        $\mathcal{D}_k \leftarrow \text{RETRIEVETOPK}(\mathcal{B}, s_t, a_t)$ ▷ Similar past experiences
        $\mathcal{H}_t \leftarrow \{(s_i, a_i)\}_{i=0}^t$ ▷ Interaction history
        $(s_{t+1}, r_{t+1}) \sim \mathcal{M}_{\text{exp}}(\cdot \mid \mathcal{H}_t, \mathcal{D}_k, \tau)$ ▷ Reasoning-based transition, Eq. (4)
        $\zeta_\tau \leftarrow \zeta_\tau \cup \{(s_t, a_t, r_{t+1})\}$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_{t+1})\}$ ▷ Co-evolving replay buffer
        **if** $\text{TERMINAL}(s_{t+1})$ **then**
          **break**
        **end if**
        $s_t \leftarrow s_{t+1}$
      **end for**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\zeta_\tau\}$ ▷ Collect trajectory for RL update
    **end for**
    Compute outcome rewards $\{r^i\}_{i=1}^n$ in group $\mathcal{G}$
    Compute value $\mathcal{V}_\tau$ for each task $\tau$ in $\mathcal{G}$ ▷ Group-based reward entropy, Eq. (7)
  **end for**
  $\theta \leftarrow \theta - \eta_\pi \nabla_\theta \mathcal{L}_{\text{RL}}(\pi_\theta; \mathcal{R})$ ▷ Policy gradient on synthetic rollouts, Eq. (1)
  $\mathcal{T}_{\text{high}} \leftarrow \{\tau \in \mathcal{T} : \mathcal{V}_\tau \text{ in top } p\%\}$ ▷ Feasible yet challenging tasks
  $\mathcal{T}_{\text{new}} \leftarrow \mathcal{M}_{\text{task}}(\mathcal{T}_{\text{high}})$ ▷ Curriculum-based task generation, Eq. (6)
  $\mathcal{T} \leftarrow \text{UPDATETASKS}(\mathcal{T}, \mathcal{T}_{\text{new}})$ ▷ Bound synthetic task proportion by $\lambda$
**end while**

### B.3 TRAINING DATA FILTERING PIPELINE

DREAMGYM involves two major stages for training RL agents: (1) training a reasoning-based experience model in the target environment to synthesize next states and rewards via explicit reasoning, and (2) interacting with this experience model to collect abundant, diverse online experience for agent RL training. Since stage (2) follows standard procedures shared by most RL frameworks and is agnostic to the downstream RL algorithm, we focus here on elaborating on the data filtering and quality-control procedures used in stage (1).

For each environment, our data filtering pipeline consists of the following three steps:

(i) Collecting offline trajectories from multiple sources. For example, in WebArena, we gather trajectories from various publicly available sources, such as the top-10 agent trace submissions on the official WebArena leaderboard, as well as the oracle trajectories in the WebArena-Lite repository, which yield more than 10K raw trajectories. Notably, we retain both successful and failed trajectories, as both encode the environment dynamics and are valuable for distilling the transition structure into the experience model. In this step, we also carefully remove all tasks and trajectories corresponding to the test set to avoid any form of test contamination.

(ii) Balancing trajectories across available training tasks as well as success and failure modes. To reduce bias during experience-model training, we first balance the trajectories collected in step (i) to ensure relatively even coverage across all training tasks. We then further filter and rebalance the remaining trajectories so that each task contains a more uniform mix of successful and failed trajectories. This mitigates overfitting to dominant failure patterns (e.g., predicting success even when actions previously failed) and enables the experience model to learn more reliable and robust environment transitions.

(iii) Ensuring diverse coverage of the action space. Based on the trajectories from the previous step, we then apply a clustering-based selection procedure to maintain coverage over diverse action sequences. This avoids overfitting the experience model to a narrow subset of actions and encourages it to learn transitions for a broad range of behaviors and accurately predict their consequences.

After applying these three stages, we obtain 3600, 5200, and 2780 high-quality trajectories for WebShop, ALFWorld, and WebArena, respectively. We then use Llama-3.3-70B as the annotator to generate a reasoning trace for each state transition in the filtered dataset, which serves as the training corpus for fine-tuning the experience model.

Overall, this simple yet effective data filtering pipeline allows us to efficiently collect large-scale offline trajectories from various sources and transform them into a diverse, balanced, and clean dataset well-suited for training a stable and reliable experience model, which can subsequently generate high-quality interaction data for RL agent training.

# C ADDITIONAL RESULTS AND ANALYSIS

## C.1 COMPARISON WITH GENERIC RL ALGORITHMS

We compare DREAMGYM against a broader set of representative generic RL baselines, including DAPO Yu et al. (2025) and Dr. GRPO Liu et al. (2025), which are developed based on GRPO to improve its efficiency and training stability, representing the most widely used RL paradigms for LLM agent post-training. As shown in Table 5, both algorithms struggle in long-horizon, real-environment settings due to infrastructure bottlenecks and low sample efficiency, which is consistent with our findings in Table 5 from the main paper. In contrast, DREAMGYM achieves substantially higher success rates on WebArena than both DAPO and Dr. GRPO. Specifically, this improvement is achieved without any real online interactions, as DREAMGYM conducts the majority of training in its synthetic environment. On AlfWorld, DREAMGYM similarly maintains strong performance, highlighting its ability to scale to multi-step embodied environments.

These results demonstrate that DREAMGYM can serve as a more scalable and effective RL training paradigm even when compared to strong general-purpose RL algorithms.

Table 5: Comparison of DREAMGYM against RL algorithms including (1) generic RL baselines (e.g. DAPO (Yu et al., 2025) and Dr. GRPO (Liu et al., 2025)) and (2) domain-engineered RL frameworks (e.g. WebAgent-R1 (Wei et al., 2025), WebRL (Qi et al., 2024), DigiRL (Bai et al., 2024)) on WebArena and AlfWorld. Metrics include averaged success rate (%), estimated time cost (hours), and the total number of real trajectories for training.

| Method | WebArena | | | AlfWorld | | |
|---|---|---|---|---|---|---|
| | Success Rate (%) | Time (h) | # Real Traj | Success Rate | Time (h) | # Real Traj |
| DAPO | 10.0 | 120 | 1200 online | 66.1 | 50 | 4800 online |
| Dr. GRPO | 12.7 | 120 | 1200 online | 70.8 | 50 | 4800 online |
| WebAgent-R1 | 44.8 | 70 | 2400 online + 9460 offline | NA | NA | NA |
| WebRL | 42.4 | 80 | 4000 online + 12200 offline | NA | NA | NA |
| DigiRL | 30.3 | 80 | 4000 online + 12200 offline | NA | NA | NA |
| DreamGym | 45.0 | 15 | 4800 offline | 70.8 | 30 | 5200 offline |
| DreamGym-S2R | 48.3 | 55 | 800 online + 4800 offline | 73.3 | 42 | 1600 online + 5200 offline |

We also evaluate DREAMGYM against domain-engineered RL frameworks, i.e., WebAgent-R1 Wei et al. (2025), WebRL Qi et al. (2024), and DigiRL Bai et al. (2024), which incorporate substantial task-specific engineering (e.g., custom intrinsic rewards, handcrafted wrappers, and reset mechanisms) designed specifically for WebArena. However, these frameworks are not transferable to other environments such as AlfWorld.

From the results shown in Table 5, we can draw the following conclusions:

(1) DREAMGYM performs comparably to these domain-engineered SOTA systems while using no real online trajectories, whereas WebAgent-R1 and WebRL each require thousands of online interactions with the real environment to achieve similar performance, despite their intrinsic reward designs and substantial engineering effort, making them expensive and difficult to scale. (2) DREAMGYM-S2R, after a modest amount of continued training in real environments, outperforms WebAgent-R1 in success rate, despite requiring far fewer real interactions and incurring dramatically lower engineering complexity. (3) Since domain-specific methods are tightly coupled to WebArena infrastructure, they cannot run on AlfWorld, while DREAMGYM generalizes seamlessly.

These findings highlight that DREAMGYM achieves SOTA-level performance without heavy domain specialization, offering far broader applicability across environments.

## C.2 COMPARISON WITH SYNTHETIC TRAINING ALGORITHMS

To further highlight DREAMGYM's design advantages, we also compare it against two concurrent frameworks that train agents in synthetic environments, Simia-RL Li et al. (2025) and UI-Simulator Wang et al. (2025), both released after our initial submission. From the results shown in Table 6, we can draw the following observations:

(1) Compared to UI-Simulator, which directly synthesizes raw DOM trees using a strong reasoning LLM, DREAMGYM achieves much higher success rates and lower time costs by synthesizing transitions in an explicit, concise, token-efficient abstract state space. This significantly reduces

inference overhead and improves the coherence of environment dynamics. (2) Compared to Simia-RL, which relies on closed-source models such as OpenAI o4-mini to generate step-wise state predictions, DREAMGYM reduces monetary cost by over 30× while achieving higher success rates. This advantage arises because DREAMGYM 's experience model is fine-tuned on domain-specific offline trajectories, enabling it to produce more consistent, in-distribution transitions that directly support effective policy improvement.

Table 6: Comparison of DREAMGYM against concurrent synthetic training frameworks on the WebArena environment. Metrics include averaged success rate (%), estimated time cost (hours), number of real-world trajectories used for training, and the monetary expense of environment service (USD) such as AWS server hosts and API key costs.

| Method | Success Rate (%) | Time (h) | # Real Traj | Expense (US Dollar) |
|---|---|---|---|---|
| Traditional | 15.7 | 120 | 1200 online | 272 |
| Simia-RL | 37.0 | 16 | 4800 offline | 460.8 |
| UI-Simulator | 19.7 | 22 | 4800 offline | 72 |
| DreamGym | 45.0 | 15 | 4800 offline | 14.4 |

### C.3 DETAILED STATISTICS OF THE ANNOTATION COSTS

During training, the primary additional cost beyond standard RL pipelines comes from annotating the reasoning traces and task-variation sets using a teacher model. Table 7 below summarizes the annotation costs for WebShop, ALFWorld, and WebArena, including both reasoning-trace and task-variation annotations. We first clarify that all reasoning-trace annotations in DREAMGYM are generated using Llama-3.3-70B-Instruct, an open-source model that can be served on a single A100 GPU; for a straightforward quantitative estimate, we convert its usage into an equivalent market price of 1 USD per 1M tokens. In particular, we do not rely on expensive closed-source LLMs such as OpenAI o4-mini, which concurrent approaches like Simia-RL (Li et al., 2025) heavily depend on for generating online interactions.

To annotate the reasoning trace for each trajectory, we provide the LLM with the full trajectory and prompt it to generate a reasoning explanation for each transition step (averaging ∼100 tokens per step; see Appendix E for the prompt template). Consequently, the annotation cost scales with the trajectory length (e.g., ALFWorld incurs slightly higher cost due to its 10–20 step average sequence length). Regardless, as shown in Table 7, the overall annotation cost for DREAMGYM remains low across all environments, requiring less than $10 in total to annotate all data needed to train the experience models.

Table 7: Detailed annotation statistics. We report the number of trajectories, reasoning-trace annotation cost (USD), and task-variation annotation cost (USD) for WebShop, ALFWorld, and WebArena. Although Llama-3.3-70B-Instruct was served locally, we estimate cost using a market rate of $1 per 1M tokens for comparability.

| Method | WebShop | AlfWorld | WebArena |
|---|---|---|---|
| #Trajectory | 3600 | 5200 | 4800 |
| Reasoning Annotation Costs | 3.6 | 7.8 | 7.2 |
| Task Variation Annotation Costs | 0.5 | 0.5 | 0.5 |

In contrast, concurrent approaches such as Simia-RL (Li et al., 2025) incur substantial ongoing costs, since they rely on SOTA models like OpenAI o4-mini to interact with the agent online throughout training, where this process must be repeated every time a new agent is trained. As shown in Table 6 above, even a modest training configuration (e.g., 150 epochs with batch size 16) results in significant inference expenses, amounting to over 30× the cost of DREAMGYM. On the contrary, DREAMGYM avoids this repeated cost by design: annotation is performed once-and-for-all in an offline setting, and the resulting fine-tuned experience model can then be reused indefinitely to generate online interactions for any agent without incurring additional inference expenses.

We also conduct an ablation comparing Llama-3.3-70B with GPT-4.1 as the annotating model, as shown in Table 8. GPT-4.1 provides only a small performance improvement over Llama-3.3-70B while costing 5× more, indicating a relatively low marginal gain. This suggests that the reasoning-trace annotation stage does not require LLMs with extremely strong reasoning capability and a powerful open-source model like Llama-3.3-70B is sufficient for the task. Accordingly, we use Llama-3.3-70B to annotate all datasets in our experiments.

Table 8: Comparison of annotation costs (USD) and averaged success rate when using Llama-3.3-70B vs. GPT-4.1 as the reasoning-trace annotator.

| Method | Llama-3.3-70B | GPT-4.1 |
|---|---|---|
| Annotation Costs | 7.2 | 36 |
| Success Rate | 45.0 | 45.6 |

### C.4 Detailed Statistics of the Costs during RL Data Collection

We provide detailed statistics of the costs incurred during the agent inference stage (i.e., RL data collection). Table 9 reports the average wall-clock time during the experience model generation per trajectory, the average number of tokens produced by the experience model per trajectory, and the total wall-clock training time across WebShop, ALFWorld, and WebArena. In addition, we include a head-to-head comparison between training agents in DreamGym and training in the real-world environment on WebArena, covering several verifiable metrics, including the average time during experience model generation per step, average time during experience model generation per trajectory, environment reset cost, and total training time, as summarized in Table 10 below.

Table 9: Detailed statistics of costs during agent RL data collection across WebShop, AlfWorld, and WebArena environments, including wall-clock training time (hours), average tokens generated by the experience model per trajectory, and wall-clock time during experience model generation per trajectory (seconds).

| Metric | WebShop | AlfWorld | WebArena |
|---|---|---|---|
| Wall-clock training time (h) | 12 | 19 | 15 |
| Avg. tokens per trajectory | 2k | 5k | 8k |
| Wall-clock time per trajectory (s) | 0.3 | 0.5 | 0.9 |

**Per-step inference.** To accommodate fast environment inference, we serve the 8B experience model using an efficient vLLM inference stack on 2×A100 GPUs. As shown in both Table 9 and Table 10, this results in very low inference latency in practice: DreamGym requires only 0.08 seconds on average to generate the reasoning trace and next-state transition for a single step. Importantly, DreamGym consumes **less than one second in total** to generate all environment states for an entire trajectory, which is negligible compared to overall GPU training time. It also consumes **fewer than 10K tokens per trajectory** (approximately 300–600 tokens per step), since the environment is synthesized in a token-efficient and sufficiently expressive abstract state space. In contrast, the real WebArena environment, even when following the official guidelines and hosting 8 concurrent instances on AWS to minimize the engineering gap, still takes **3.4 seconds per step** due to substantial overhead such as HTTP requests, page rendering, and resource fetching. This does not include the additional human intervention often required to resolve issues such as pages getting stuck on login screens or unstable server responses, all of which further degrade the speed and reliability of real-environment execution.

**Environment reset.** Moreover, as recommended by WebArena's official documentation, we manually reset all environment Dockers after every 50 tasks to avoid cross-task interference, where each reset takes 300 seconds on average. Although automated scripts are possible, we found them unstable and less operational in practice. On the contrary, DreamGym does not incur any comparable overhead, and resetting the environment simply involves resetting the prompt, which is instantaneous.

**Sampling failures.** In addition, we also report the failure rate during trajectory sampling, which is the proportion of trajectories where at least one state transition fails. Such failures frequently occur in WebArena due to infrastructure instability, e.g., network timeouts, page breakdowns, and cross-task interference, an open issue also explicitly acknowledged in their repository. In contrast, while DREAMGYM also incurs certain failures (e.g., occasional repetition patterns in model outputs), they occur far less frequently as model-based transitions are more reliable and controllable. Importantly, unlike the real environment where a single failed transition invalidates the entire trajectory (requiring resampling from the beginning), DREAMGYM can simply resample from the failed state, significantly improving efficiency and fault tolerance.

Table 10: Detailed breakdown of time costs on WebArena, including average time per step (seconds), average time per trajectory (seconds), environment reset time (seconds), total training time (hours), and trajectory sampling failure rate.

| Method | Avg Time per Step (s) | Avg Time per Traj (s) | Reset Time (s) | Total Training Time (h) | Sampling Failure Rate (%) |
|---|---|---|---|---|---|
| DreamGym | 0.08 | 0.9 | 0 | 15 | 1.7 |
| Real Environment | 3.4 | 54.4 | 300 | 120 | 19.0 |

Eventually, these factors accumulate into large differences in overall training time. As shown in Table 10, DREAMGYM completes a full training run in 15 hours, with the majority of time spent on GPU training rather than environment inference. In contrast, training with the real WebArena environment requires over 120 hours, where environment interaction dominates the total cost. Moreover, simply hosting the WebArena instances on AWS incurs a non-negligible monetary expense (approximately $272 for 120 hours), as reported in Table 6.

### C.5 ABLATION EXPERIMENTS ON REPLAY BUFFER

We conducted an ablation study evaluating the effect of the synthetic-data ratio $\beta$ of the replay buffer on both end-to-end agent performance and synthetic state quality. Consistent with the settings in figure 4, state quality is assessed using GPT-4o as a judge, including key criteria such as hallucination, consistency, diversity, and informativeness, where a higher score indicates the state is more consistent, diverse, and factual with respect to the real counterpart. We present the evaluation results on WebArena in Table 11 below. When $\beta = 0.3$, the agent achieves the highest success rate while maintaining the same state-quality score as $\beta = 0$ (only real data, where no error accumulation during self-training is guaranteed), suggesting that introducing up to 30% synthetic data does not cause additional hallucination or observable error accumulation. Even at $\beta = 0.5$, the state quality score remains high and does not significantly affect the agent's end-to-end learning effectiveness.

These findings demonstrate that our buffer update and retrieval mechanisms effectively maintain the quality of the synthetic transitions in the replay buffer, while still allowing them to be challenging and co-evolve with the agent's improving policy.

Table 11: Ablation on the ratio of synthetic data $\beta$ in the replay buffer. We report end-to-end success rate on WebArena and the average state-quality score evaluated by GPT-4o as judge.

| Method | Success Rate | State Judge Score |
|---|---|---|
| $\beta = 0$ | 42.0 | 1.6 |
| $\beta = 0.1$ | 43.7 | 1.6 |
| $\beta = 0.3$ | 45.0 | 1.6 |
| $\beta = 0.5$ | 42.9 | 1.5 |

### C.6 GENERALIZATION AND LEARNING TRANSFERABILITY.

The results in figure 3 *Middle* highlight the strong generalization and cross-domain transferability of the policy trained in DREAMGYM. Notably, (1) when trained on WebShop (Yao et al., 2022), the policy generalizes to WebArena (Zhou et al.) and surpasses SFT models directly trained there, and (2) when trained on WebArena, it similarly transfers back to WebShop with superior performance than the SFT as well. This generalization suggests that DREAMGYM learns within an abstract meta-representation space, enabling the agent to learn domain-agnostic behavioral priors rather than memorizing task-specific patterns. However, (3) when the domain gap becomes too large, such as transferring from web-based environments (WebShop/WebArena) to ALFWorld (Shridhar et al.), the performance drops significantly, indicating the limits of current meta-representations.

# D THEORETICAL ANALYSIS

In this section, we analyze how policies trained in the synthetic environments of DREAMGYM can provably improve performance in real environments. We show that, under mild assumptions, performance guarantees can be established by optimizing learning-centric signals of the experience model, such as reward accuracy and domain consistency, rather than strict fidelity metrics like state reconstruction error.

## D.1 PROVABLY POLICY IMPROVEMENT IN REAL ENVIRONMENTS TRAINED WITH SYNTHETIC EXPERIENCES

DREAMGYM trains LLM agents using a reasoning-based experience model $\mathcal{M}\mathrm{exp}$, which interacts with the agent and induces a synthetic MDP $\widehat{\mathcal{M}}$. For brevity, we use $\widehat{\mathcal{M}}$ to denote any such synthetic environment, including $\mathcal{M}_{\mathrm{exp}}$, which is defined in the abstract textual state space, as stated in section 4.1. The learned policy is then evaluated in the real environment $\mathcal{M}$, projected into the same abstract space for comparison. We show that, under standard trust-region policy update assumptions, a policy optimized in $\widehat{\mathcal{M}}$ is guaranteed to also achieve policy improvement in the real environment $\mathcal{M}$.

**Theorem 1** (Policy Improvement $J$ in Real Environment via Synthetic Experiences). *Let the real MDP be $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, the synthetic MDP induced by $\mathcal{M}_{\mathrm{exp}}$ be $\widehat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \widehat{P}, \widehat{R}, \gamma)$, discount be $\gamma \in (0, 1)$, and let rewards be bounded $R, \widehat{R} \in [0, R_{\max}]$ with $V_{\max} := R_{\max}/(1-\gamma)$. Assume one-step experience-model errors*

$$\varepsilon_R \;:=\; \sup_{s,a} \big| R(s,a) - \widehat{R}(s,a) \big|, \qquad \varepsilon_P \;:=\; \sup_{s,a} \mathrm{TV}\big(P(\cdot|s,a), \widehat{P}(\cdot|s,a)\big), \tag{8}$$

*and a trust-region update $\pi \to \pi'$ obtained* by optimizing in $\widehat{\mathcal{M}}$ *with per-state KL radius $\sup_s D_{\mathrm{KL}}(\pi'(\cdot|s) \,\|\, \pi(\cdot|s)) \le \delta$, as enforced by the soft KL penalty in PPO and GRPO. Hence*

$$J_{\mathcal{M}}(\pi') - J_{\mathcal{M}}(\pi) \;\ge\; \underbrace{\frac{1}{1-\gamma}\, \mathbb{E}_{s\sim d^{\pi}_{\widehat{\mathcal{M}}},\, a\sim\pi'(\cdot|s)}\big[A^{\pi}_{\widehat{\mathcal{M}}}(s,a)\big]}_{\text{synthetic surrogate gain in } \mathcal{M}_{\mathrm{exp}}} - \underbrace{\frac{4\gamma}{(1-\gamma)^2}\, V_{\max}\, \delta}_{\text{trust-region penalty}} - \underbrace{2\left(\frac{\varepsilon_R}{1-\gamma} + \frac{2\gamma R_{\max}}{(1-\gamma)^2}\varepsilon_P\right)}_{\text{experience model error}} \tag{9}$$

*In particular, if the synthetic surrogate gain exceeds the two penalties, then $J_{\mathcal{M}}(\pi') \ge J_{\mathcal{M}}(\pi)$.*

Specifically, (1) the *synthetic surrogate gain* denotes the agent's performance improvement when trained and evaluated within the synthetic environment provided by the experience model $\mathcal{M}_{\mathrm{exp}}$. (2) The *trust-region penalty* corresponds to the KL radius $\delta$ constraint, which is softly enforced by PPO or GRPO. (3) The *experience-model error* measures how well $\mathcal{M}_{\mathrm{exp}}$ preserves *learning-relevant signals* of the original environment for agent knowledge acquisition including two key components: (a) the **faithfulness of feedback** ($\varepsilon_R$), i.e., how accurately reward signals reflect real outcomes, and (b) the **domain consistency of state transitions** ($\varepsilon_P$), i.e., how well state space distributions align with the dynamics from the original environment.

Notably, these two error terms align with our design insights in section 4.1: **the synthetic environment need only provide domain-consistent transitions and correct, retrospective learning signals, without having to clone the original environment at the raw state level**. In practice, both $\varepsilon_R$ and $\varepsilon_P$ can be made very small even when $\mathcal{M}_{\mathrm{exp}}$ is trained with minimal trajectory data annotated with explicit reasoning traces.

*Proof of Theorem 1.* We first decompose the policy improvement in the real environment through the synthetic environment:

$$J_{\mathcal{M}}(\pi') - J_{\mathcal{M}}(\pi) = \big(J_{\widehat{\mathcal{M}}}(\pi') - J_{\widehat{\mathcal{M}}}(\pi)\big) + \big(J_{\mathcal{M}}(\pi') - J_{\widehat{\mathcal{M}}}(\pi')\big) - \big(J_{\mathcal{M}}(\pi) - J_{\widehat{\mathcal{M}}}(\pi)\big). \tag{10}$$

By Lemma 1, each of the two policy discrepancy terms $\big|J_{\mathcal{M}}(\cdot) - J_{\widehat{\mathcal{M}}}(\cdot)\big|$ is at most $\Delta_{\mathrm{model}}$, hence

$$J_{\mathcal{M}}(\pi') - J_{\mathcal{M}}(\pi) \;\ge\; \big(J_{\widehat{\mathcal{M}}}(\pi') - J_{\widehat{\mathcal{M}}}(\pi)\big) - 2\Delta_{\mathrm{model}}. \tag{11}$$

It remains to lower bound improvement inside the synthetic environment. Using the standard trust-region bound (Schulman et al., 2015), which is enforced in practice by PPO and GRPO via a per-state KL radius $\delta$, we have

$$J_{\widehat{\mathcal{M}}}(\pi') - J_{\widehat{\mathcal{M}}}(\pi) \;\ge\; \frac{1}{1-\gamma}\, \mathbb{E}_{s\sim d^{\pi}_{\widehat{\mathcal{M}}},\, a\sim\pi'(\cdot|s)}\big[A^{\pi}_{\widehat{\mathcal{M}}}(s,a)\big] - \frac{4\gamma}{(1-\gamma)^2}\, V_{\max}\, \delta. \tag{12}$$

Combining these two terms yields the inequality in Theorem 1, which completes the proof. $\square$

**Lemma 1** (Multi-turn experience synthesis error bound)**.** *For any policy $\pi$, if*

$$\varepsilon_R = \sup_{s,a} |R(s,a) - \widehat{R}(s,a)|, \qquad \varepsilon_P = \sup_{s,a} \mathrm{TV}\big(P(\cdot|s,a), \widehat{P}(\cdot|s,a)\big), \tag{13}$$

*then*

$$\big|J_{\mathcal{M}}(\pi) - J_{\widehat{\mathcal{M}}}(\pi)\big| \;\le\; \Delta_{\text{model}} \;\coloneqq\; \frac{\varepsilon_R}{1-\gamma} + \frac{2\gamma R_{\max}}{(1-\gamma)^2}\,\varepsilon_P. \tag{14}$$

*Proof.* We first compare the Bellman operators of the real and synthetic environments. For any bounded value function $V$,

$$(T_\pi V)(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\big[R(s,a) + \gamma\,\mathbb{E}_{s'\sim P(\cdot|s,a)} V(s')\big], \tag{15}$$

and let $\widehat{T}_\pi$ be the same expression with $(R, P)$ replaced by $(\widehat{R}, \widehat{P})$. Thus for any bounded value function $V$, the operator difference is bounded as

$$\|T_\pi V - \widehat{T}_\pi V\|_\infty \le \sup_{s,a} |R(s,a) - \widehat{R}(s,a)| + \gamma \sup_{s,a} \Big|\mathbb{E}_{s'\sim P(\cdot|s,a)} V(s') - \mathbb{E}_{s'\sim \widehat{P}(\cdot|s,a)} V(s')\Big| \tag{16}$$

$$\le \varepsilon_R + 2\gamma \|V\|_\infty \varepsilon_P, \tag{17}$$

which is derived by simply using the definitions of $\varepsilon_R, \varepsilon_P$ and the variational characterization of TV. Now apply this bound to $V = V^\pi_{\mathcal{M}}$ and add–subtract:

$$\|V^\pi_{\mathcal{M}} - V^\pi_{\widehat{\mathcal{M}}}\|_\infty = \|T_\pi V^\pi_{\mathcal{M}} - \widehat{T}_\pi V^\pi_{\widehat{\mathcal{M}}}\|_\infty \tag{18}$$

$$\le \|T_\pi V^\pi_{\mathcal{M}} - \widehat{T}_\pi V^\pi_{\mathcal{M}}\|_\infty + \|\widehat{T}_\pi V^\pi_{\mathcal{M}} - \widehat{T}_\pi V^\pi_{\widehat{\mathcal{M}}}\|_\infty \tag{19}$$

$$\le \varepsilon_R + 2\gamma V_{\max}\varepsilon_P + \gamma \|V^\pi_{\mathcal{M}} - V^\pi_{\widehat{\mathcal{M}}}\|_\infty. \tag{20}$$

By rearranging the contraction term into the left side, we have

$$(1-\gamma)\|V^\pi_{\mathcal{M}} - V^\pi_{\widehat{\mathcal{M}}}\|_\infty \le \varepsilon_R + 2\gamma V_{\max}\varepsilon_P. \tag{21}$$

Hence

$$\|V^\pi_{\mathcal{M}} - V^\pi_{\widehat{\mathcal{M}}}\|_\infty \le \tfrac{1}{1-\gamma}\big(\varepsilon_R + 2\gamma V_{\max}\varepsilon_P\big). \tag{22}$$

Finally, since $J_{\mathcal{E}}(\pi) = \mathbb{E}_{s_0\sim\mu}[V^\pi_{\mathcal{E}}(s_0)]$, we obtain

$$\big|J_{\mathcal{M}}(\pi) - J_{\widehat{\mathcal{M}}}(\pi)\big| = \Big|\mathbb{E}_{s_0\sim\mu}\big[V^\pi_{\mathcal{M}}(s_0) - V^\pi_{\widehat{\mathcal{M}}}(s_0)\big]\Big| \tag{23}$$

$$\le \|V^\pi_{\mathcal{M}} - V^\pi_{\widehat{\mathcal{M}}}\|_\infty \tag{24}$$

$$\le \frac{\varepsilon_R}{1-\gamma} + \frac{2\gamma R_{\max}}{(1-\gamma)^2}\varepsilon_P \tag{25}$$

$$=: \Delta_{\text{model}}. \tag{26}$$

This indicates that the gap of agent performance between real and synthetic environments depends only on reward accuracy and domain consistency errors, rather than on strict fidelity metrics such as state reconstruction error, etc. $\square$

# E DREAMGYM PROMPTS

## E.1 WEBSHOP

**Experience model reasoning step annotation | `WebShop`**

**System Prompt:** You are an expert in web navigation and e-commerce environments, specializing in providing actionable guidance for state transitions of an experience model that simulates the environment dynamics.

**User Prompt:** You are synthesizing environment state transition plans for training world models in webshopping tasks. You are provided with a task `instruction`, a `flag` indicating whether the trajectory is successful, and a `trajectory` $\{(s_i, a_i)\}_{i=1}^N$ of the environment state and the corresponding agent action at each step.

**Task Context**:
Task: {`instruction`} Success: {`flag`}
**Trajectory Steps:**
`" ".join(["Step: ` $\{i\}$`, Environment State: ` $\{s_i\}$`, Action: ` $\{a_i\}$`"]` $_{i=1}^{N}$`)`
**Your Task**:

- **Task Tutorial**: A high-level guidance of how the environment should transit step-by-step to interact with the agent under the given task instruction. It should highlight the critical steps that the agent should perform in order for the environment to transit to the final successful state.
- **State Transition Plans**: For each step, first analyze whether the agent's action is likely to success or fail based on the task tutorial (e.g. the search query is too vague or too specific, or the agent clicks the wrong product), and then provide a concise reasoning trace describing how the environment should transition given the current state and action.

**CRITICAL**: You MUST generate exactly one transition plan for each environment step provided and your `state_transitions` array must contain exactly `len(env_step_ids)` entries, one for each `step_id`. For product listing pages, the state transition plan should mention some actionable details such as the number of products shown on this page, whether this page should contain the target product given the agent's action. Focus on actionable guidance for training the experience model. Keep responses concise and practical.

**Response Format**: json { "task_tutorial":
{"Overall Plan": "A one-sentence high-level guidance of how the environment should transit step-by-step to interact with the agent under the given task instruction.",
"Success Mode": "Describe the critical steps that the agent should perform to succeed in the task, where the environment should correspondingly transit to the successful state. Summarize in one sentence.",
"Failure Mode": "Describe the typical failure mode the agent should avoid, where the environment should correspondingly transit to the failed state once the agent performs the action. Summarize in one sentence." },
"state_transitions": [{
"step_id": 0,
"transition_plan": "Analyze whether the agent's action is good or bad based on the next state and overall task tutorial, and a corresponding plan for how environment should respond to this action."}
...
] }

**Task variation dataset construction | `WebShop`**

**System Prompt:** You are an expert in e-commerce task design and AI training data curation.

**User prompt:** You are an expert in e-commerce task design. I will give you an original web shopping `task instruction` and several `candidate variations` of this task. Your job is to select the most challenging yet feasible variation that would be good to train an AI agent to acquire the skills of shopping for the given `product`.

**Original Task**: {`task instruction`}

**Product Information**:

1. Category: {`product_info['category']`}
2. Product Name: {`product_info['name']`}
3. Available Attributes: {`','.join(product_info['attributes'])`}

**Candidate Variations**:{`candidates variations`}

**Criteria for selection**:

- **Challenging but Feasible**: The task should be more specific or complex than the original, but still achievable, so as to strengthen the agent's capabilities for shopping for the given product.
- **High Quality**: The instruction should be clear, grammatically correct, and realistic.
- **Meaningful Variation**: The changes should make the task meaningfully different (not just trivial changes).
- **Realistic**: The combination of attributes, options, and price should make sense for the product category.

**Please respond with**:

1. The number of your selected variation 1- (`len({candidate variations})`).
2. A brief explanation (1-2 sentences) of why this variation is the most challenging and high-quality.

**Format your response as**:
SELECTION: [number]
REASONING: [explanation]

**Agent Prompt Template | `WebShop`**

You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to:
{`task_description`}.

Prior to this step, you have already taken {`step_count`} step(s). Below are the most recent {`history_length`} observations and the corresponding actions you took:
{`action_history`}
You are now at step {`current_step`} and your current observation is:
{`current_observation`}.
Your admissible actions of the current situation are:
[ {`available_actions`} ].

Now it's your turn to take one action for the current step. You should first reason step-by-step about the current situation, then think carefully which admissible action best advances the shopping goal. This reasoning process MUST be enclosed within `<think> </think>` tags.
Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

## E.2 ALFWORLD

**Task variation dataset construction | `ALFWorld`**

**System Prompt:** You are an expert in embodied task design and AI training data curation for interactive embodied environments.

**User prompt:** You are an expert in embodied task design. I will give you a feasible `task instruction` for an embodied agent and several `candidate variations` of this task. Your job is to select the most challenging yet feasible variation that would be good to train an AI agent to acquire generalizable embodied reasoning skills.
**Original Task**: {`task instruction`}
**Environment Context**:

1. Room Type: {`env_info['room']`}
2. Objects Present: {`','.join(env_info['objects'])`}
3. Containers/Surfaces: {`','.join(env_info['locations'])`}

**Candidate Variations**: {`candidate variations`}
**Criteria for selection**:

- **Challenging but Feasible**: The variation should add complexity (e.g., more objects, constraints, or multi-step actions) without being impossible.
- **High Quality**: Clear, grammatical, and realistic in the ALFWorld context.
- **Meaningful Variation**: Should involve non-trivial differences in *action type*, *target object*, or *target location*.
- **Realistic**: The variation must be consistent with ALFWorld's embodied environment dynamics (e.g., no placing a fridge on a lamp).

**Please respond with**:

1. The number of your selected variation 1-(`len({candidate variations})`).
2. A brief explanation (1-2 sentences) of why this variation is the most challenging and high-quality.

**Format your response as**:
SELECTION: [number]
REASONING: [explanation]

**Agent Prompt Template | `ALFWorld`**

You are an expert agent operating in the ALFRED Embodied Environment.
Your task is to:
{`task_description`}

Prior to this step, you have already taken {`step_count`} step(s). Below are the most recent {`history_length`} observations and the corresponding actions you took: {`action_history`}.
You are now at step {`current_step`} and your current observation is: {`current_observation`}.
Your admissible actions of the current situation are: [{`admissible_actions`}].

Now it's your turn to take an action. You should first reason step-by-step about the current situation. This reasoning process MUST be enclosed within `<think> </think>` tags. Once you've finished your reasoning, you should choose an admissible action for current step and present it within `<action> </action>` tags.

### E.3 WEBARENA

**Task variation dataset construction | `WebArena`**

| | |
|---|---|
| **System Prompt:** | You are an expert in designing realistic, diverse, and challenging web interaction tasks for AI agents. |
| **User prompt:** | I will provide you with several seed `WebArena task instructions`. Your job is to generate new task variations from each seed. The variations should keep the **same** general action type (e.g., search, filter, upvote, navigate, purchase, delete) but **differ** in target, constraints, or context, making them realistic, challenging, and meaningfully different.<br>**Seed Instructions**: {`list of seed instructions`}<br>**Requirements for variations**:<br>• **Action Consistency**: Preserve the same type of action as the seed task.<br>• **Meaningful Differences**: Change the entities, filters, domains, time ranges, or constraints so the new task is distinct but natural.<br>• **Challenging but Feasible**: The variation should slightly increase reasoning or constraint complexity, but remain solvable.<br>• **High Quality**: Grammatically correct, clear, and realistic web tasks.<br>**Please respond with**:<br>For each seed instruction, generate [K] new task variations. Format your response as:<br>SEED: [original instruction]<br>VARIATIONS:<br>1. [variation 1]<br>2. [variation 2]<br>...<br>**Example**:<br>SEED: List products from living room furniture category by descending price.<br>VARIATIONS:<br>1. List products from bedroom furniture category by ascending price.<br>2. Show me the most expensive three dining tables available online.<br>3. Find discounted sofas under $500 in the living room furniture category. |

**High-quality task variation selection | `WebArena`**

| | |
|---|---|
| **System Prompt:** | You are an expert in interactive web task design and AI training data curation. |
| **User prompt:** | You are an expert in web environment task design. I will give you an original WebArena `task instruction` and several `candidate variations` of this task. Your job is to select the most challenging yet feasible variation that would be good to train an AI agent to acquire generalizable skills in web interaction.<br>**Original Task**: `{task instruction}`<br>**Candidate Variations**: `{candidate variations}`<br>**Criteria for selection**:<br>• **Challenging but Feasible**: The variation should require slightly more reasoning, precision, or constraints than the original, but still be solvable by a web agent.<br>• **High Quality**: Clear, grammatical, and realistic within the web environment.<br>• **Meaningful Variation**: Keep the same *action type* (e.g., search, navigate, sort, submit, upvote, purchase) as the original, but change the context, target, or condition.<br>• **Realistic**: The task should reflect plausible web interactions a user might request.<br>**Please respond with**:<br>1. The number of your selected variation 1-(`len({candidate variations})`).<br>2. A brief explanation (1-2 sentences) of why this variation is the most challenging and high-quality.<br>**Format your response as**:<br>SELECTION: [number]<br>REASONING: [explanation] |

**AX-tree state mapping prompt | `WebArena`**

**System Prompt:**

You are an agent tasked with extracting and refine a subset of the webpage's observations based on the content of the page and user instructions. Perform the following tasks based on the provided [Information source], including user instructions, interaction history, and the AXTree observation at the current time step. First, provide high-level reasoning for the next action by analyzing the provided information. Second, extract relevant webpage elements based on your high-level reasoning.

**User prompt:**

[General instructions]
You are currently on the {`domain_info`} website. Your task is to generate a **Reasoning** and a **Refined observation** based on the provided inputs.
First, review the **User instruction** and **History of interactions** and, then, generate the **Reasoning**. Analyze the progress made so far, and provide a rationale for the next steps needed to efficiently accomplish the user instruction on the {`domain_info`} website.
Second, refine the **Webpage observation at the current time step** into a **Refined observation**. Extract a subset of the webpage observation (e.g., chart, table, menu items) that contains necessary information for completing the user instruction, and explain the extracted elements. Ensure that the information on the elements (e.g., numeric element ID) is correctly included.
Please follow the format in the [Reasoning & Refinement example] carefully.

[Information source]
**User instruction**: {`user instruction`}
**History of interactions**:{`interaction history`}
**Webpage observation at the current time step**:{`AXTree observation`}

[Reasoning & Refinement example]
**Abstract example**
Here is an abstract version of the answer, describing the content of each tag. Make sure you follow this structure and format strictly, but replace the content with your own answer:
`<reasoning>`
Think step by step. Based on the **User instruction**, **History of interaction**, and **AXTree observation at the current time step**:

- Provide a high-level description of the **AXTree observation at the current time step.**
- Based on the **User instruction** and **History of interaction** track your progress and provide your reasoning on the next action needed to accomplish the **User instruction**

Ensure that: Structure your reasoning concisely and follow the following format strictly:
`<content_description>` High-level description of current page state (max 2 sentences)`</content_description>` `<agent_progress>` What has been accomplished so far (max 1 sentence)`</agent_progress>` `<next_action_analysis>` What should happen next and why (max 1 sentence)`</next_action_analysis>`
`</reasoning>`
`<extraction>`
Based on your reasoning, identify the elements (e.g., buttons, text fields, static text, table row, chart) to focus on. Then, explain the semantics and functionalities of each extracted element. Ensure that: You do not alter the structure of the AXTree observation. You extract the element ID (id in [ ]) accurately without any errors. When extracting chart or table, you must extract the entire chart or table to avoid any confusion or loss of information. Unless necessary, try not to extract url or non-semantic identifiers which is not informative for the agent actions. All the elements you extract should be actionable and discard irrelevant elements. Please follow the following format and do not provide any other text besides the element list.
[ELEMENT_ID] TYPE:DESCRIPTION
[ELEMENT_ID] TYPE:DESCRIPTION
...
[ELEMENT_ID] TYPE:DESCRIPTION
(Extract 3-10 most relevant actionable elements only)
`</extraction>`

**Agent Prompt Template | `WebArena`**

**System Prompt:** You are an agent trying to solve a web task based on the content of the page anda user instructions. You can interact with the page and explore. Each time you submit an action it will be sent to the browser and you will receive a new page.

**User prompt:** **Instructions**
Review the current state of the page and all other information to find the best possible next action to accomplish your goal. Your answer will be interpreted and executed by a program, make sure to follow the formatting instructions.
**User instruction**: `{user instruction}`
**History of interactions**:`{interaction history}`
**Refined observation of current step**: Reasoning `{plan}`
**Focused AXTree observation**: `{rep_observation}`
**Action space**: 13 different types of actions are available.

- noop(wait_ms: float = 1000)
  1. Description: Do nothing, and optionally wait for the given time (in milliseconds).
  2. Examples: noop(),noop(500)
- ...

To save space, please refer to E.3.1 for the full list of actions.

**Remark:** Only a single action can be provided at once. Example:`fill('a12', 'example with "quotes"')` Multiple actions are meant to be executed sequentially without any feedback from the page. Don't execute multiple actions at once if you need feedback from the page.

**Abstract Example**
Here is an abstract version of the answer with description of the content of each tag. Make sure you follow this structure, but replace the content with your answer:
`<think>`
Think step by step. If you need to make calculations such as coordinates, write them here. Describe the effect that your previous action had on the current content of the page.
`</think>`
`<action>`
One single action to be executed. You can only use one action at a time.
`</action>`

**Concrete Example**
Here is a concrete example of how to format your answer. Make sure to follow the template with proper tags:

```
<think>
My memory says that I filled the first name and last name, but I can't see any content in the form. I need to explore different ways to fill the form. Perhaps the form is not visible yet or some fields are disabled. I need to replan. </think>
<action>
fill('a12', 'example with "quotes"')
</action>
```

### E.3.1 ACTION SPACE OF WEBARENA

- **noop** `(wait_ms: float = 1000)`
  1. Description: Do nothing, and optionally wait for the given time (in milliseconds).
  2. Examples: `noop()`; `noop(500)`
- **send_msg_to_user** `(text: str)`
  1. Description: Send a message to the user. You should send a short answer as a message and do not ask questions through message.
  2. Examples: `send_msg_to_user('the city was built in 1751.')`; `send_msg_to_user('Yes')`; `send_msg_to_user('No')`; `send_msg_to_user('31112')`; `send_msg_to_user('Yoshua Bengio')`
- **scroll** `(delta_x: float, delta_y: float)`
  1. Description: Scroll horizontally and vertically. Amounts in pixels, positive for right or down scrolling, negative for left or up scrolling. Dispatches a wheel event.
  2. Examples: `scroll(0, 200)`; `scroll(-50.2, -100.5)`
- **fill** `(bid: str, value: str)`
  1. Description: Fill out a form field. It focuses the element and triggers an input event with the entered text. It works for `<input>`, `<textarea>` and `[contenteditable]` elements.
  2. Examples: `fill('237', 'example value')`; `fill('45', 'multi-line example')`; `fill('a12', 'example with "quotes"')`
- **select_option** `(bid: str, options: str | list[str])`
  1. Description: Select one or multiple options in a `<select>` element. You can specify option value or label to select. Multiple options can be selected.
  2. Examples: `select_option('48', 'blue')`; `select_option('48', ['red', 'green', 'blue'])`
- **click** `(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`
  1. Description: Click an element.
  2. Examples: `click('51')`; `click('b22', button='right')`; `click('48', button='middle', modifiers=['Shift'])`
- **dblclick** `(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`
  1. Description: Double click an element.
  2. Examples: `dblclick('12')`; `dblclick('ca42', button='right')`; `dblclick('178', button='middle', modifiers=['Shift'])`
- **hover** `(bid: str)`
  1. Description: Hover over an element.
  2. Examples: `hover('b8')`
- **press** `(bid: str, key_comb: str)`
  1. Description: Focus the matching element and press a combination of keys. It accepts the logical key names that are emitted in the `keyboardEvent.key` property of the keyboard events: `Backquote, Minus, Equal, Backslash, Backspace, Tab, Delete, Escape, ArrowDown, End, Enter, Home, Insert, PageDown, PageUp, ArrowRight, ArrowUp, F1 - F12, Digit0 - Digit9, KeyA - KeyZ, etc.` You can alternatively specify a single character you'd like to produce such as `"a"` or `"#"`. Following modification shortcuts are also supported: `Shift, Control, Alt, Meta`.
  2. Examples: `press('88', 'Backspace')`; `press('a26', 'Control+a')`; `press('a61', 'Meta+Shift+t')`
- **focus** `(bid: str)`
  1. Description: Focus the matching element.
  2. Examples: `focus('b455')`
- **clear** `(bid: str)`
  1. Description: Clear the input field.
  2. Examples:`clear('996')`
- **drag_and_drop** `(from_bid: str, to_bid: str)`
  1. Description: Perform a drag & drop. Hover the element that will be dragged. Press left mouse button. Move mouse to the element that will receive the drop. Release left mouse button.
  2. Examples: `drag_and_drop('56', '498')`
- **upload_file** `(bid: str, file: str | list[str])`
  1. Description: Click an element and wait for a "filechooser" event, then select one or multiple input files for upload. Relative file paths are resolved relative to the current working directory. An empty list clears the selected files.
  2. Examples: `upload_file('572', 'my_receipt.pdf')`; `upload_file('63', ['/home/bob/Documents/image.jpg', '/home/bob/Documents/file.zip'])`

### E.4 Experience Model Judge

**Judge Prompt for Evaluating Experience Models**

You are an expert judge for scoring the quality of a predicted state transition in a WebShop environment simulator.

**You are given**:
- Current state (before the action)
- The agent action
- Predicted next state (after the action)

**Your task**:
1) Evaluate the predicted next state on **four rubrics**, each scored **0**, **1**, or **2**.
2) Provide brief step-by-step reasoning for **each rubric**.
3) Output a **valid JSON** object with the rubric scores and the total (sum of the four rubrics). Do not include extra fields.

**General rules**:
- Base your judgment **only** on the provided inputs; do not assume hidden context.
- Use integers only (**0/1/2**) for rubric scores.
- If an action is invalid or should not change the page, correct behavior may include a no-op with an explicit failure/empty-result signal.
- Be concise but specific in your reasoning (1–3 sentences per rubric). —

**# # # Rubrics (0/1/2) with anchors:**

1) **Causal State Consistency** | *Question:* Is the predicted next state both logically consistent with the prior state **and** causally grounded in the agent's action semantics (e.g., click → detail page, pagination → new results, search → updated listings, back → prior view)?
   - **2**: Coherent and action-appropriate; all expected updates appear with no contradictions.
   - **1**: Mostly consistent, but has minor logical or semantic gaps.
   - **0**: Inconsistent or not causally linked to the action.

2) **Diversity & State Variation** | *Question:* Is there a meaningful, non-degenerate change from the prior state (when change is expected)?
   - **2**: Substantive, coherent differences (new results, updated filters, changed details).
   - **1**: Minimal or superficial change.
   - **0**: No meaningful change, or incoherent jump.

3) **Informativeness** | *Question*: Is the predicted state rich, relevant, and internally coherent (e.g., listings with meaningful attributes; filters aligned with content)?
   - **2**: Detailed, relevant, and coherent information.
   - **1**: Some useful details, but sparse or partially incoherent.
   - **0**: Uninformative, irrelevant, or incoherent.

4) **Hallucination & Failure Feedback** | *Question*: When the action is invalid or yields no results, does the state reflect an appropriate failure/empty-result signal instead of hallucinating success?
   - **2**: Correctly signals failure or success as appropriate, no hallucination.
   - **1**: Partial/ambiguous handling of failure.
   - **0**: Hallucinates success or ignores failure.

—

**### Step-by-step Evaluation (use this structure):**
1. **Causal State Consistency:** <your reasoning> **Score:** 0/1/2
2. **Diversity & State Variation:** <your reasoning> **Score:** 0/1/2
3. **Informativeness**: <your reasoning> **Score:** 0/1/2
4. **Hallucination & Failure Feedback**: <your reasoning> **Score:** 0/1/2

—

**# # # Final JSON Output:**

Output a single valid JSON object. Replace angle brackets with integers only.

```
{"rubric_scores": {
"causal_consistency": <0|1|2>, "diversity": <0|1|2>,
"informativeness": <0|1|2>, "hallucination": <0|1|2> }}
```