SEAGym: An Evaluation Environment for Self-Evolving LLM Agents

Anonymous ACL submission

Abstract

Self-evolving LLM agents improve mainly by changing their surrounding harness: prompts, memories, tools, skills, middleware, and other persistent control surfaces. Existing evaluations often reduce this process to isolated task scores or a single sequential curve, obscuring whether an update produces reusable improvement, overfits recent tasks, increases cost, or harms older behavior. We introduce SEAGYM, an evaluation environment for measuring agent-supplied harness updates across training, validation, test, replay, and cost records. SEAGYM turns Harbor-compatible benchmarks into dynamic self-evolution task sources with train batches, frozen update-validation, held-out in-domain and transfer tests, replay diagnostics, and auditable snapshot records. Instantiating SEAGYM on Terminal-Bench 2.0 and HLE, we compare ACE, TF-GRPO, and AHE under a shared epoch/batch protocol. The results show that these evaluation views provide complementary signals about the evolution process: frequent updates may fail to improve held-out performance, useful intermediate snapshots may collapse later, and source diversity and model backend can affect harness reliability.

1 Introduction

LLM-based agents are no longer fixed systems at deployment time. They can store experience, revise prompts, write memories, add skills, change tool-use routines, or edit runtime configuration. These components form an *agent harness*: the persistent, model-external control layer around the base model that mediates task interaction, tool use, memory, and execution (Yang et al., 2024; Wang et al., 2025a; Li et al., 2026; Pan et al., 2026). In this paper, a *self-evolving agent* is an LLM-based agent that uses task experience to update this persistent harness state, and then reuses the updated state on later tasks.

Self-evolution can occur through different processes. An agent may update itself while solving a single task, use what it learned on one task for the next task, or repeatedly train on a set of tasks across multiple rounds. Methods also differ in what they update. Text-centered methods revise prompts, reflections, instructions, or experience libraries (Shinn et al., 2023; Madaan et al., 2023; Agrawal et al., 2026; Cai et al., 2025). Memory-based and skill-based methods build reusable memories, skills, workflow traces, or knowledge bases (Wang et al., 2023, 2024; Xu et al., 2025; Tang et al., 2025). Harness-level methods change broader execution structure, including tools, middleware, sub-agents, workflows, or project files (Zhuge et al., 2024; Zhang et al., 2025; Yuan et al., 2025a; Lin et al., 2026). These differences can be expressed with the same interaction loop: the agent acts in tasks, observes trajectories and feedback, updates persistent state, and acts again. A self-evolution benchmark should therefore model the process, not only the final task score. This is why self-evolution is naturally modeled as an RL environment: the environment provides task exposure, feedback, and evaluation views, while the agent controls the policy and update rule.

This view exposes several research questions. Does the update improve future performance, or only adapt to the task that produced it? Does it extract reusable behavior, or memorize narrow patterns from recent tasks? Does it transfer from source tasks to held-out tasks and new domains, or overfit to one benchmark? Does it preserve older abilities while improving new ones? Answering these questions requires a configurable dynamic evaluation setting, not only a task list or a sequential score curve. Task exposure, feedback visibility, update timing, validation views, transfer tests, and replay checks must be specified independently so that adaptation, generalization, transfer, and for-

getting can be measured separately.

Current benchmarks make this hard to study. Most agent benchmarks are designed for static evaluation: each task is an isolated episode, the agent state is reset, and the score measures one fixed agent (Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Mialon et al., 2024; Patil et al., 2025; Yao et al., 2024). This removes the state carryover that self-evolving agents are meant to use. Sequential and lifelong evaluations keep the agent in a task stream and are closer to deployment (Jiang et al., 2026; Zheng et al., 2025). However, a single stream curve still mixes many effects. Later tasks may be easier, more similar to earlier tasks, or solved with more computation. An update may help recent tasks while hurting older skills, or improve training tasks without transferring to held-out tasks. Thus the main gap is not only the lack of harder tasks; it is the lack of an evaluation environment that can model the self-evolution process and evaluate adaptation, generalization, transfer, and forgetting.

To mitigate this gap, we introduce SEAGYM, an evaluation environment for self-evolving LLM agents. SEAGYM turns existing static benchmarks into task sources inside a dynamic self-evolution loop: it exposes train batches, lets agents update through their native methods, records frozen snapshots, and evaluates those snapshots on views that are not used as update evidence. It represents different self-evolution processes with explicit schedule parameters, including state reset, task reuse, batch size, and update timing, so single-task adaptation, online transfer, and epoch-based batch learning can be studied under one environment. For execution, SEAGYM builds on Harbor, a framework for running agent evaluations and RL environments in containerized task settings (Harbor Framework Team, 2026). The two systems are complementary: Harbor provides task execution, environments, verifiers, and parallel jobs, while SEAGYM turns static benchmark tasks into train batches, validation views, final tests, and transfer settings for self-evolution studies. The current paper instantiates this path with Terminal-Bench 2.0 (Merrill et al., 2026) and HLE (Phan et al., 2025), and separates task rollout from method update so diverse self-evolving agents can be connected through thin wrappers while preserving their native update rules.

Our contributions are:

- We introduce SEAGYM, an evaluation environment that converts Harbor-compatible static benchmarks into dynamic self-evolution task sources and connects diverse baselines through a thin rollout/update interface.
- We formulate self-evolution as an RL environment over agent snapshots, with configurable schedules for single-task adaptation, online transfer, and epoch-based batch learning, and with held-out views for validation, final testing, transfer, replay, and diagnostics.
- Through extensive experiments on Terminal-Bench 2.0 and HLE, we validate the effectiveness of the SEAGYM design and use it to study current self-evolving mechanisms, revealing useful patterns about update quality, held-out generalization, transfer, forgetting, cost, schedule choice, source diversity, and model backends.

2 Related Work

Agent harnesses. Agent harnesses denote the non-parametric mechanisms that mediate a model's interaction with tasks, including tool interfaces, memory, orchestration, runtime isolation, feedback, and policy controls. Early agent work studied reasoning–acting loops and API/tool use (Yao et al., 2023; Schick et al., 2023; Patil et al., 2023; Qin et al., 2024); newer work shows that tool documentation, agent-computer interfaces, and memory management are themselves performance-critical design variables (Yuan et al., 2025b; Yang et al., 2024; Xiong et al., 2025). Recent platform and protocol work further systematizes the harness as a composable layer for execution, interoperability, observability, verification, and runtime enforcement (Wang et al., 2025a; Ehtesham et al., 2025; Wang et al., 2026; Li et al., 2026). The same shift appears in production practice, where OpenAI, Anthropic, and LangChain describe harness engineering in terms of structured environments, feedback loops, durable execution, middleware, and agent-legible state (OpenAI, 2026a; Anthropic, 2026; LangChain, 2025, 2026). Pan et al. (2026) make this separation explicit by expressing harness modules in natural language, supporting inspection and ablation of the non-model agent state. This perspective makes harness state a natural

target for adaptation: prompts, memories, tools, middleware, and policies can persist beyond a single episode and affect performance on subsequent tasks.

Continual learning and self-evolution. Continual learning studies systems that face a sequence of tasks or data distributions, where learning from new experience can improve future behavior but may also interfere with previously acquired capabilities. Its central evaluation concerns—adaptation, transfer, retention, replay, and forgetting—are therefore relevant to self-evolving agents (Parisi et al., 2019; Robins, 1995; Kirkpatrick et al., 2017; Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2019; van de Ven and Tolias, 2019). Self-evolving LLM agents extend this setting from parameter learning to agent-system learning. One direction improves the underlying model or behavior policy with supervised tuning, reinforcement learning, self-play, process-level rewards, or tool-use training (Yuan et al., 2025c; Kumar et al., 2024; Setlur et al., 2025; Choudhury, 2025; Wang et al., 2025b; Feng et al., 2025). A second direction treats the harness itself as the object of evolution: environmental feedback can revise prompts, memories, tool interfaces, workflows, communication structure, or middleware without retraining the base model (Fang et al., 2025; Zhuge et al., 2024; Zhang et al., 2025; Wang et al., 2024; Yuan et al., 2025a; Xu et al., 2025; Tang et al., 2025). This distinction matters for evaluation because harness updates are persistent, method-specific, and may be applied inside the same execution loop that produces the evidence used for updating. Recent methods differ not only in what component they change, but also in whether they rely on reflection, verifier feedback, rollout comparison, or search, and whether the update is applied within a task, between tasks, after a batch, or over repeated epochs (Zhang et al., 2026; Agrawal et al., 2026; Cai et al., 2025; Lin et al., 2026). As a result, reported gains are hard to compare without a shared protocol that separates train exposure, validation evidence, held-out transfer, replay, and reset assumptions.

Agent benchmarks. Agent benchmarks provide the environments in which harness behavior becomes observable. Recent surveys cover evaluations of agent capabilities and task settings such as planning, tool use, memory, software repair, terminal operation, scientific reasoning, web and desktop interaction, function calling, and tool-agent-user workflows (Yehudai et al., 2026; Jimenez et al., 2024; Merrill et al., 2026; Phan et al., 2025; Zhou et al., 2024; Xie et al., 2024; Mialon et al., 2024; Patil et al., 2025; Yao et al., 2024). These benchmarks mark a shift from text-only scoring toward interactive environments with executable actions, state changes, and task-specific verifiers. Their standard protocol, however, still evaluates a fixed agent on independent episodes; persistent harness state is reset or not supported. Benchmarks explicitly targeting self-evolution or lifelong agent learning remain scarce. SEA-Eval evaluates agents over constructed sequential task streams and separates genuine evolution from token-consumption artifacts, while LifelongAgentBench builds skill-grounded lifelong-learning tasks across interactive database, operating-system, and knowledge-graph environments (Jiang et al., 2026; Zheng et al., 2025). These efforts leave open a complementary direction: converting existing agent benchmarks into environments for both evolution and assessment, with support for controlling task exposure, feedback visibility, update timing, held-out validation, transfer tests, retention checks, and diagnostics across different self-evolution methods.

3 Method: SEAGym

3.1 Problem Formulation

We model a self-evolution run as an MDP-style evaluation process

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \rho), \tag{1}$$

where each state $s_t \in \mathcal{S}$ contains the current agent snapshot, the schedule position, and the available task context. An agent snapshot is

$$A_t = (M, H_t), \tag{2}$$

where M denotes the fixed base model and immutable runtime components, and H_t denotes the mutable harness state: prompts, memories, skills, experience libraries, tools, middleware, project files, or other model-external control surfaces. At step t, the evaluation environment presents a task batch B_t. The agent solves the tasks, producing trajectories $\mathcal{T}_t$ and receiving feedback F_t according to the benchmark's visibility policy. It then applies its own update rule

$$H_{t+1} = U(H_t, B_t, \mathcal{T}_t, F_t), \tag{3}$$

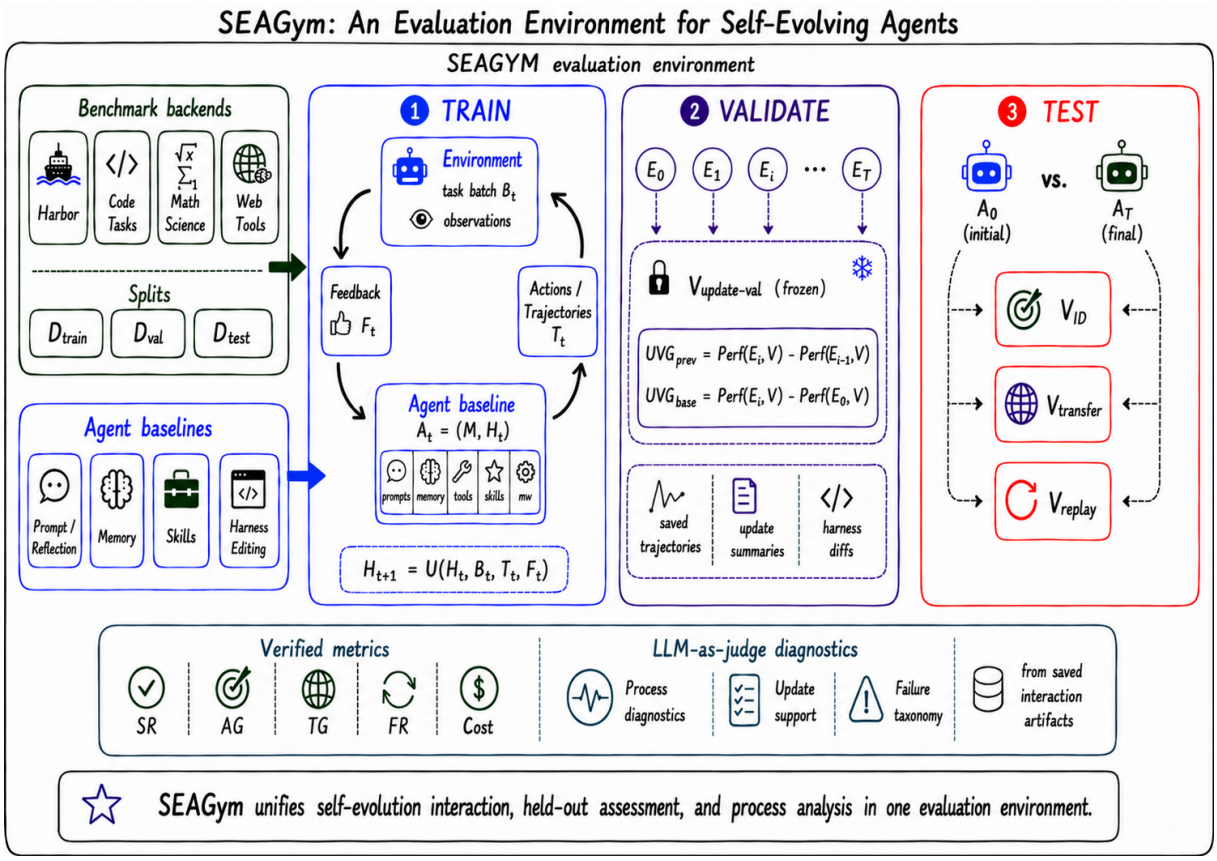

Figure 1: Overview of SEAGYM. The environment constructs train batches, collects trajectories and feedback, lets the self-evolving agent update its own state, and records evaluation points as frozen snapshots. Snapshot quality is measured with frozen update-validation, final held-out tests, replay diagnostics, and cost metrics.

which, together with task execution and verifier feedback, induces the transition from s_t to s_{t+1}. SEAGYM specifies the observed environment: task streams, feedback visibility, schedules, snapshot boundaries, and held-out assessment views. It leaves the policy and update rule to each self-evolving agent and asks whether later snapshots improve beyond the update-bearing tasks.

3.2 Evolution Schedule

SEAGYM represents online, single-task, batch, and epoch-based self-evolution settings through schedule parameters rather than separate benchmark formats. The main experiments use persistent state, repeated train batches, train-batch updates, and epoch-end frozen update-validation assessment. Appendix A gives the schedule table, saved records, and epoch/batch loop in Algorithm 1.

3.3 Data Splits and Evaluation Views

SEAGYM keeps the base data split conventional: D_{train}, D_{val}, and D_{test}. Update-validation, replay, transfer, and diagnostics are defined as experiment-specific evaluation views, not as permanent split categories. This avoids encoding distribution relationships into a task itself: a task can be in-domain in one experiment and transfer in another, depending on the selected train source. Table 1 lists the core pools and views.

The main protocol evaluates native self-updates without using validation views to choose or modify them. Train batches produce update evidence, while validation and test views are reserved for frozen assessment. In the paper setting, update-validation runs at epoch end on the same frozen $V_{\text{update-val}}$, and final assessment compares A_0 and A_T on ID, transfer, and replay views.

3.4 Benchmark and Baseline Integration

SEAGYM is designed to integrate both benchmark suites and self-evolving methods with minimal new code. On the benchmark side, it reuses Harbor as the task runner (Harbor Framework Team, 2026). Harbor retains task definitions, environments, verifiers, and parallel exe-

ML role	SEAGym pool/view	Purpose
Update exposure	D_{train}	visible tasks used to produce trajectories, feedback, and agent updates
Update assessment	$V_{\text{update-val}} \subset D_{\text{val}}$	hidden, frozen view used to assess intermediate snapshots
ID generalization	$V_{\text{ID}} \subset D_{\text{test}}$	final held-out test from the source distribution
Transfer	$V_{\text{transfer}} \subset D_{\text{test}}$	final held-out test under domain or distribution shift
Retention	V_{replay}	scheduled or final check for forgetting or regression

Table 1: Core pools and evaluation views in SEAGYM. Base manifests remain conventional train/validation/test splits; self-evolution-specific assessments are experiment views.

cution, while SEAGYM adds the self-evolution schedule: train batches, frozen validation views, final held-out views, snapshot timing, and metric computation. This turns static benchmark tasks into a dynamic self-evolution environment without rewriting the benchmark. In the current paper, the same Harbor-facing path turns Terminal-Bench 2.0 and HLE Math/Physics into source train/validation/test views, and HLE CS/AI plus Engineering into held-out transfer views. The environment exposes only the task information needed for solving, while keeping private evaluation metadata such as reference outputs, split membership, and held-out view labels out of the agent input.

On the method side, SEAGYM separates rollout from update. A rollout component runs tasks under the current harness state and returns a trajectory batch. An update component consumes that trajectory batch and applies the method's native self-evolution rule. This decomposition gives prompt, memory, skill, textual-optimization, context-update, and harness-editing methods a common interface while leaving each method's update semantics intact. In practice, connecting a new baseline only requires a thin wrapper that converts SEAGYM trajectory batches to the method's native update input and saves the resulting harness state. We use the same boundary in reusable integration checklists and skill templates: benchmark authors follow Harbor task/adapter templates, and baseline authors implement the rollout/update wrapper. Additional interface details are in Appendix D. Figure 1 summarizes this flow.

3.5 Process Diagnostics with LLM Judges

Verified task rewards are the primary source of benchmark scores, but they do not fully explain the evolution process. SEAGYM can therefore attach offline LLM-as-judge diagnostics to saved trajectories, update summaries, and harness diffs. These judges produce process annotations, such as evidence-support scores, overfitting warnings, failure tags, or update-quality labels linked to evaluation points. They support qualitative analysis and comparisons among self-evolution mechanisms, while primary metrics remain based on verifier scores, held-out views, replay checks, and measured cost.

3.6 Metrics

SEAGYM computes metrics from saved records so results can be audited and recomputed without rerunning environments. Given verifier score $r(A, x) \in [0, 1]$, it reports

$$\text{Perf}(A, D) = \frac{1}{|D|} \sum_{x \in D} r(A, x), \tag{4}$$

$$\text{SR}(A, D) = \frac{1}{|D|} \sum_{x \in D} \mathbb{I}[r(A, x) = 1]. \tag{5}$$

For update-validation view $V = V_{\text{update-val}}$ and evaluation point E_i:

$$\text{UVG}_i^{\text{prev}} = \text{Perf}(E_i, V) - \text{Perf}(E_{i-1}, V), \tag{6}$$

$$\text{UVG}_i^{\text{base}} = \text{Perf}(E_i, V) - \text{Perf}(E_0, V). \tag{7}$$

For final ID, transfer, and replay views D_I, D_T, D_R:

$$\text{IDG} = \text{Perf}(A_T, D_I) - \text{Perf}(A_0, D_I), \tag{8}$$

$$\text{TG} = \text{Perf}(A_T, D_T) - \text{Perf}(A_0, D_T), \tag{9}$$

$$\text{FR} = \max(0, \text{Perf}(A_0, D_R) - \text{Perf}(A_T, D_R)). \tag{10}$$

We also report token usage, tool calls, wall-clock time, and cost reduction when available. Main result tables use domain-level macro averages by default so that larger task groups do not dominate cross-domain conclusions. Appendix E summarizes the metric table and additional logging details.

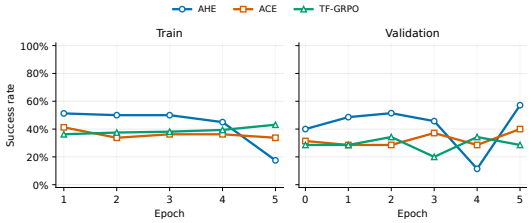

Figure 2: Baseline learning curves for AHE, ACE, and TF-GRPO.

4 Experiments

4.1 Experimental Setup

We use an epoch-based batch setting over 80 source train tasks from Terminal-Bench 2.0 and HLE text-only Math/Physics, with 35 source validation tasks, 55 source test tasks, and 80 HLE CS/AI and Engineering transfer tasks (Merrill et al., 2026; Phan et al., 2025). Unless stated otherwise, runs use five epochs and train batch size 20. The main comparison evaluates ACE, TF-GRPO, and AHE under DeepSeek-V4-Flash; the ablations use AHE for batch size, source diversity, and cross-model transfer (Zhang et al., 2026; Cai et al., 2025; Lin et al., 2026; DeepSeek-AI, 2026; OpenAI, 2026b; Z.AI, 2026). Appendix Tables 7 and 8 list the full settings.

4.2 Baseline Results

Figure 2 and Table 2 compare the three completed DeepSeek-V4-Flash runs. Appendix Figures 4 and 7 provide source-group and batch-index breakdowns.

AHE is the only baseline that improves validation, ID, and OOD together. ACE improves validation but transfers weakly to held-out views, while TF-GRPO changes state at every update and spends the most rollout tokens but loses ID/OOD performance. Thus, validation gain and update activity alone are insufficient; the benchmark separates update activity, held-out generalization, and cost.

4.3 Training Fix and Forgetting

We next replay the source train set with each AHE epoch-end snapshot. Figure 3 reports the train replay grid; Appendix Figure 12 provides the corresponding success, forgetting, and fix-rate curves.

The final snapshot solves 43/80 train-replay tasks, compared with 34/80 for the initial snapshot, but the trajectory is not monotonic. The epoch-4 snapshot drops to 6/80 and produces many rollout errors before the final snapshot recovers. This shows why snapshot-level diagnostics are needed: an initial-versus-final table would miss a damaging intermediate harness regression.

4.4 Effect of Batch Size

We vary the AHE train batch size while keeping the source views and epoch count fixed. Table 3 summarizes validation, ID, cost, and update status; Appendix Figures 5 and 8 provide curve-level breakdowns.

Batch size is non-monotonic: batch 20 is the only setting with large positive validation and ID gains, while batch 10 and batch 80 regress. This suggests that AHE is limited by evidence inspection, not only by update frequency. Most update tokens are spent reading rollout records; batch 80 dilutes attention across too many trajectories, while batch 10 updates frequently on small, high-variance evidence. Neither larger batches nor more frequent updates are automatically better.

4.5 Source Diversity

We compare the main mixed-source AHE run with an HLE-only run of the same train size. Table 4 reports final reliability and the best available intermediate snapshot; Appendix Figures 6 and 9 provide curve-level breakdowns.

The HLE-only run reaches a useful intermediate snapshot, but its final snapshot collapses on validation, ID, and OOD. This suggests that a single benchmark can drive the harness toward benchmark-specific local optima. The mixed-source run also passes through a bad intermediate state, so diversity does not prevent harmful updates. Its advantage is recovery: Terminal-Bench exposes tool, environment, and execution failures, while HLE exposes reasoning failures, giving later updates more varied evidence for repairing a broken harness.

4.6 Cross-Model Transfer

Finally, we swap evolved AHE harnesses across rollout models. Table 5 reports OOD transfer for DeepSeek-evolved and GLM-evolved harnesses; Appendix C.1 reports the GPT-5.4 continuation trajectory.

The OOD gains are highest when the evolved harness is executed by the same model family that evolved it. The DeepSeek-evolved harness improves OOD under DeepSeek rollout, but loses transfer under GLM and GPT-5.4 rollout. The

Agent	V_0	$V_\star$	UVG	ID_0	$\text{ID}_\star$	IDG	OOD_0	$\text{OOD}_\star$	TG	Rollout tok./task	Update tok./update	Updates
AHE	40.0	57.1	+17.1	40.0	49.1	+9.1	22.5	28.8	+6.3	1.46M	3.91M	20/20 changed
ACE	31.4	40.0	+8.6	34.5	36.4	+1.8	20.0	21.2	+1.3	1.55M	–	8/20 changed
TF-GRPO	28.6	34.3	+5.7	36.4	34.5	-1.8	21.2	15.0	-6.3	1.69M	0.16M	20/20 changed

Table 2: Baseline results. Success rates are percentages, gains are percentage points, and token costs are normalized per task or update.

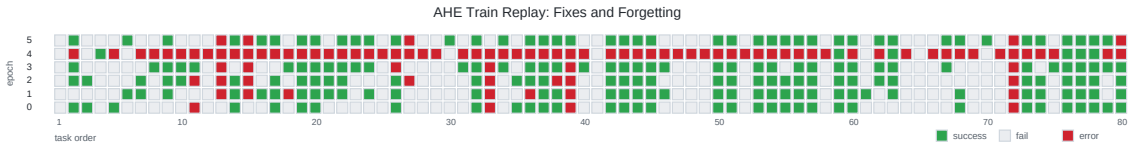

Figure 3: AHE train replay grid. Green cells are successful trials, gray cells are failed trials, and red cells are rollout errors.

Batch	V_0	V_T	UVG	ID_0	ID_T	IDG	Rollout tok./task	Update tok./update	Updates
10	37.1	22.9	-14.3	38.2	23.6	-14.5	0.79M	3.13M	39/40 changed
20	40.0	57.1	+17.1	40.0	49.1	+9.1	1.46M	3.91M	20/20 changed
40	37.1	40.0	+2.9	41.8	43.6	+1.8	1.09M	3.36M	10/10 changed
80	42.9	25.7	-17.1	41.8	25.5	-16.4	1.07M	3.57M	5/5 changed

Table 3: Effect of batch size. Success rates are percentages, gains are percentage points, and token costs are normalized per task or update.

Source	Final snapshot				Selected epoch				
	V_T	ID_T	IDG_T	OOD_T	$\text{Epoch}_\star$	$\text{ID}_\star$	$\text{IDG}_\star$	$\text{OOD}_\star$	$\text{TG}_\star$
Terminal-Bench + HLE	57.1	49.1	+9.1	28.8	5 (final)	49.1	+9.1	28.8	+6.3
HLE only	0.0	0.0	-22.0	0.0	3	47.3	+7.3	25.0	+3.8

Table 4: Source diversity. Final columns measure end-of-training reliability; selected-epoch columns report the best intermediate snapshot. Success rates are percentages and gains are percentage points.

Evolved harness	Rollout model	ID	OOD A_0	OOD evolved	OOD gain
DeepSeek-evolved	DeepSeek	–	22.5	28.8	+6.3
DeepSeek-evolved	GLM	49.1	26.3	22.5	-3.8
DeepSeek-evolved	GPT-5.4	41.8	33.8	28.8	-5.0
GLM-evolved	GLM	–	22.5	27.5	+5.0
GLM-evolved	DeepSeek	45.5	26.3	17.5	-8.8
GLM-evolved	GPT-5.4	47.3	33.8	33.8	0.0

Table 5: Cross-model OOD transfer. Scores are percentages; OOD gain compares the evolved harness to the same rollout model's A_0 result.

GLM-evolved harness improves OOD under GLM rollout, is neutral under GPT-5.4 rollout, and drops sharply under DeepSeek rollout. Thus, AHE learns model-adapted harness changes rather than a model-independent harness.

5 Conclusion

SEAGYM evaluates self-evolving agents through the behavior of their evolving harnesses across train exposure, frozen update-validation, held-out tests, replay, and cost records. The experiments show that these views are complementary: AHE improves validation, ID, and OOD together, while ACE and TF-GRPO expose weaker transfer or costly update behavior. Replay diagnostics reveal that a run can end improved while passing through a damaging intermediate harness state. The ablations further show that evolution is constrained by rollout-evidence inspection: neither larger batches nor more frequent updates are automatically better, source diversity improves recovery from bad updates, and cross-model transfer is strongest when a harness is executed by the model family that evolved it.

Limitations

SEAGYM depends on stable tasks, reliable verifiers, and reproducible execution environments. The current experiments are also expensive because snapshot replay, OOD transfer, and cross-model rollout exchange require repeated evaluation of saved harness states. Some method wrappers expose incomplete update artifacts or token accounting, so missing coverage is reported rather than imputed. Finally, the present cross-model results are incomplete for GPT-5.4 final ID/OOD evaluation and should be read as process evidence until those held-out jobs are complete.

Ethics Statement

SEAGYM is intended as an evaluation framework. The benchmark should avoid exposing hidden verifier details or private oracle artifacts to agents. If future task domains include real user data, web data, or proprietary repositories, dataset construction must follow licensing, privacy, and anonymization requirements.

References

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*.

Anthropic. 2026. Harness design for long-running application development. https://www.anthropic.com/engineering/harness-design-long-running-apps.

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Yong Mao, Ke Li, and Xing Sun. 2025. Training-free group relative policy optimization. *arXiv preprint arXiv:2510.08191*.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. 2019. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*.

Sanjiban Choudhury. 2025. Process reward models for LLM agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*.

DeepSeek-AI. 2026. DeepSeek V4 Preview Release. https://api-docs.deepseek.com/news/news260424.

Usama Ehtesham, Salam Dib, Sufian Almajali, Tiago Peixoto, Jay Bhattacharya, Anupam Singla, and Themistoklis Diamantopoulos. 2025. A survey of agent interoperability protocols: Model context protocol (MCP), agent communication protocol (ACP), agent-to-agent protocol (A2A), and agent network protocol (ANP). *arXiv preprint arXiv:2505.02279*.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. 2025. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. ReTool: Reinforcement learning for strategic tool use in LLMs. *arXiv preprint arXiv:2504.11536*.

Harbor Framework Team. 2026. Harbor: A framework for evaluating and optimizing agents and models in container environments.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Zhou, Jiaqing Liang, and Yanghua Xiao.

2026. SEA-eval: A benchmark for evaluating self-evolving agents beyond episodic assessment. *arXiv preprint arXiv:2604.08988*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

LangChain. 2025. Agent frameworks, runtimes, and harnesses - oh my! https://www.langchain.com/blog/agent-frameworks-runtimes-and-harnesses-oh-my.

LangChain. 2026. How middleware lets you customize your agent harness. https://blog.langchain.com/how-middleware-lets-you-customize-your-agent-harness/.

Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Jianyang Gu, Yingqiang Ge, Weijie Xu, Xi Fang, Xiang Xu, Tianchen Zhao, Youngeun Kim, Tianyang Wang, Jihun Hamm, Smita Krishnaswamy, Jun Huan, and Chandan K. Reddy. 2026. Agent harness engineering: A survey. https://picrew.github.io/LLM-Harness/. Under review.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. 2026. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*.

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

William Merrill and 1 others. 2026. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*.

OpenAI. 2026a. Harness engineering: Leveraging Codex in an agent-first world. https://openai.com/index/harness-engineering/.

OpenAI. 2026b. OpenAI API model documentation: GPT-5.4. https://platform.openai.com/docs/models/gpt-5.4.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. 2026. Natural-language agent harnesses. *arXiv preprint arXiv:2603.25723*.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*.

Long Phan and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2025. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *International Conference on Learning Representations*.

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

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, Hao Cheng, Chi Wang, and Wangchunshu Zhou. 2025. Agent KB: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Gido M. van de Ven and Andreas S. Tolias. 2019. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. 2026. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, and 1 others. 2025a. The OpenHands software agent SDK: A composable and extensible foundation for production agents. *arXiv preprint arXiv:2511.03690*.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. 2025b. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Tuo Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yiheng Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*.

Guangzhi Xiong, Qiao Jin, Zhizheng Zhang, Xiao Lu, Zhiyong Wang, Meng Ma, Xunzhu Wang, Yiyang Wang, Yikai Liu, Huaxiu Sun, Fei Wang, Zhiyong Liu, and Chenyan Liu. 2025. How memory management impacts LLM agents: An empirical study of experience-following behavior. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. Survey on evaluation of LLM-based agents. *arXiv preprint arXiv:2503.16416*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. 2025a. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. *arXiv preprint arXiv:2406.14228*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2025b. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 951–972.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2025c. Self-rewarding language models. In *International Conference on Machine Learning*.

Z.AI. 2026. GLM-5.1 model documentation. `https://docs.z.ai/guides/llm/glm-5.1`.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Language agents as optimizable graphs. In *International Conference on Machine Learning*.

A Additional Method Details

This appendix follows the structure of the main paper. Appendix A expands the method and protocol details from Section 3. Appendix B describes the benchmarks, baselines, concrete settings, and recorded compute budget used by Section 4. Appendix C contains supplementary learning curves and replay diagnostics. Appendix D and Appendix E summarize implementation and metric records needed to audit the results.

A.1 Evolution Schedule Fields

The main paper uses an epoch-based batch setting, but the same protocol can instantiate other self-evolution schedules. The experiment configuration records the following schedule fields:

- **state carryover**: whether the harness state is reset between tasks or persists across task exposure;
- **task reuse**: whether the update-bearing task stream is one pass or repeated across epochs;
- **train size**: the number of tasks selected from the source train pool for each epoch;
- **batch size**: the number of tasks exposed before each train-batch update;
- **number of epochs**: the number of traversals over the selected train set;
- **updates per batch**: the number of rollout-update attempts on each train batch;
- **assessment timing**: the boundaries at which frozen snapshots are evaluated.

Schedule parameter	Paper setting
Agent state	persistent baseline state
Training epochs	`num_epochs` over the train split
Batch size	`batch_size=k`
Update repeats	`num_updates_per_batch`
Evaluation schedule	epoch-end update-validation; final test

Table 6: Evolution schedule parameters used by SEAGYM.

Algorithm 1 SEAGYM Epoch/Batch Evaluation

Require: task index, train/val/test splits, schedule, seed
Require: self-evolving baseline with rollout policy and update rule U
Construct train batches $\{B_t\}$ and select $V_{\text{update-val}}$, replay views, and final views
Record initial snapshot $A_0 = (M, H_0)$ as E_0
for epoch $= 1, \ldots, N$ **do**
  for train batch B_t **do**
    Run rollout agent on B_t and collect $(\mathcal{T}_t, F_t)$
    Update harness state: $H_{t+1} \leftarrow U(H_t, B_t, \mathcal{T}_t, F_t)$
    Save update summary and optional checkpoint
  end for
  Freeze current agent as E_i and evaluate on $V_{\text{update-val}}$ ▷ epoch-end assessment
end for
Evaluate A_0 and A_T on final ID, transfer, and replay views ▷ held-out assessment

In the default paper setting, state is persistent, tasks are reused across epochs, updates occur after train batches, and frozen update-validation assessment occurs at epoch end.

A.2 Saved Evaluation Views

Base manifests contain only train, validation, and test task ids. Before a scored run, the data module selects and saves:

- train batches from the train split;
- $V_{\text{update-val}}$ from the validation split;

- final ID views from held-out source test tasks;
- final transfer views from held-out target-domain test tasks;
- replay or diagnostic views when enabled.

The saved task ids are part of the run artifact so that metric computation can be audited without relying on in-memory sampling state.

B Experimental Details

B.1 Benchmarks

The paper experiments use two Harbor-backed benchmark sources. Terminal-Bench 2.0 provides executable command-line and software-engineering tasks with containerized environments and verifiers (Merrill et al., 2026). HLE provides expert-level question-answering tasks (Phan et al., 2025); we use text-only Math and Physics tasks as source tasks, and text-only CS/AI plus Engineering tasks as held-out transfer tasks. Harbor provides the task execution substrate, including environments, parallel jobs, trial artifacts, and verifier result files (Harbor Framework Team, 2026). SEAGYM does not copy benchmark definitions; it stores task ids, stable attributes, split membership, schedule records, snapshot records, and normalized metric inputs.

B.2 Baselines

We connect self-evolving methods through the rollout/update interface described in Section 3. ACE evolves persistent context from task traces (Zhang et al., 2026). TF-GRPO uses grouped rollout evidence to update an experience/context store without model-weight training (Cai et al., 2025). GEPA performs reflective textual prompt evolution (Agrawal et al., 2026); the current paper records GEPA as a diagnostic run because its native update path mostly produced unchanged updates or contaminated final evaluations in the observed run window. AHE edits a broader agent harness, including prompts, middleware, memory, and project files, using observability from previous rollouts (Lin et al., 2026). For all methods, task execution remains Harbor-backed and the method wrapper preserves native update semantics as much as possible.

B.3 Paper Experiment Settings

Table 7 lists the full setting used by the main experiments and ablations. The main text keeps only the short version needed to understand the results.

Setting	Value
Task source	Terminal-Bench 2.0 plus HLE text-only Math and Physics for source training, update-validation, and in-domain testing.
Source split sizes	80 train tasks, 35 validation tasks, and 55 in-domain test tasks.
Transfer test source	HLE text-only CS/AI and Engineering, used only as held-out C transfer tests.
Transfer test size	80 total tasks.
Train feedback	Train tasks provide task trajectories, public errors, verifier rewards, and method-visible artifacts according to the task interface.
Validation feedback	$V_{\text{update-val}}$ is frozen before scored runs and is used only for epoch-end assessment.
Final test feedback	In-domain and C transfer tests are run only from frozen snapshots and are not fed back to the update rule.
State behavior	Agent harness state persists across train batches and epochs. Final evaluation compares the initial snapshot A_0 with the final snapshot A_T.
Main schedule	Five epochs, train batch size 20, and epoch-end frozen update-validation.
Update attempts	All completed main-table runs use one SEAGym update attempt per train batch. Method-native rollouts, such as TF-GRPO train-only grouped evidence, are recorded separately from SEAGym update attempts.
Rollout attempts	Validation and final views use single-attempt evaluation. TF-GRPO preserves train-only grouped rollout evidence for its native update rule.
Main methods	ACE, TF-GRPO, GEPA, and AHE. Each method uses its native harness update rule and saved update artifacts.
Main model backend	DeepSeek-V4-Flash for the main method comparison; exact provider metadata is recorded with the run artifacts.
Batch-size ablation	AHE with batch sizes 10, 40, and 80, compared against the main batch size 20 schedule.
Source-choice ablation	AHE on HLE-only Math and Physics source tasks, compared with the combined Terminal-Bench plus HLE source setting.
Cross-model ablation	AHE with GLM-5.1 and GPT-5.4, compared against the DeepSeek-V4-Flash AHE run.
Execution backend	Harbor-backed task execution with SEAGym snapshot, schedule, metric, and artifact records.
Runtime budget	Scored paper configs use task timeout 1800 seconds and concurrency 16 unless overridden by a run-specific config.
Primary metric	Task success rate on frozen validation and test views, reported as a percentage.
Derived metrics	Update-validation gain, final in-domain gain, transfer gain, forgetting, cost, and token usage when available; gain values are percentage-point differences.
Diagnostics	Update status, setup failures, verifier failures, provider errors, runtime corruption, replay or regression probes when enabled, and harness-diff summaries when available.

Table 7: Full experiment setting for the main paper runs and ablations. Run-specific overrides are recorded in the corresponding config and artifact files.

B.4 Experiment-Specific Configurations

Table 8 lists the concrete run configuration behind each experiment subsection. All run configs live under `runs/paper_main/configs/` with filenames matching the run ids in the saved artifacts. All scored training runs use seed 42, Harbor/E2B execution, task timeout 1800 seconds, verifier timeout 600 seconds, and concurrency 16 unless otherwise noted. The main Terminal-Bench + HLE split is `paper_main_terminal_hle_seed42_80_35_55`; the HLE-only split is `hle_math_physics_seed42_80_35_50`; the OOD transfer split is `hle_ood_testset_seed42_0_0_80`.

Paper section	Training runs	Source / schedule	Model backend	Extra evaluations and reported view
Baseline Results	AHE main, ACE main, and TF-GRPO main	Terminal-Bench + HLE; 80/35/55 split; five epochs; batch size 20; one update per train batch	DeepSeek v4 Flash for rollout and update where applicable	ACE and TF-GRPO use best-validation snapshot evaluations for ID/OOD. AHE uses the final snapshot plus AHE OOD evaluate jobs. The table reports validation, ID, OOD, token cost, and update status.
Training Fix and Forgetting	AHE main	Same as AHE main; fixed saved snapshots at initialization, epochs 1–4, and the final epoch	DeepSeek v4 Flash	All fixed snapshots are evaluated on the 80 source train tasks. The grid reports success, failure, and rollout-error status by original train exposure order.
Effect of Batch Size	AHE batch-size sweep: 10, 20, 40, and 80	Terminal-Bench + HLE; five epochs; batch sizes 10/20/40/80; one update per train batch	DeepSeek v4 Flash	Final ID evaluations use each run's final snapshot. OOD is not included for this ablation because batch-size OOD evaluations are not complete for all batch sizes.
Source Diversity	Mixed-source AHE and HLE-only AHE	Main source: Terminal-Bench + HLE 80/35/55. HLE-only source: HLE Math/Physics 80/35/50. Both use five epochs and batch size 20	DeepSeek v4 Flash	Main source uses final ID/OOD evaluations. HLE-only reports final validation/ID and the best-validation OOD snapshot because the final HLE-only snapshot collapses.
Cross-Model Transfer	AHE evolved with DeepSeek-V4-Flash, GLM-5.1, and GPT-5.4	Terminal-Bench + HLE; five epochs; batch size 20; one update per train batch	DeepSeek-V4-Flash, GLM-5.1, and GPT-5.4, one model per training run	Cross-backend snapshot evaluations swap evolved AHE snapshots and rollout backends on ID/OOD views. Clean final gains are reported only for completed and uncontaminated final evaluations; contaminated or incomplete runs are diagnostic.

Table 8: Experiment-specific configurations for the results in Section 4. Each run artifact records the resolved JSON config, split, model, backend, and snapshot metadata.

B.5 Training Runs, Tokens, and Recorded Runtime

Table 9 summarizes the paper-facing training runs currently represented in the saved artifacts. Token counts are read from normalized SEAGym metric records. Rollout tokens count task-execution records, and update tokens count SEAGym baseline-update records when the native method exposes token usage. Runtime is the sum of unique Harbor-reported task-job runtimes recovered from saved task results. It is therefore a recorded task-execution runtime, not a complete end-to-end wall-clock measurement; native update wall time and external queueing delays were not recorded consistently across methods.

Run	Baseline / rollout	Model	Batch	Epochs	Updates	Task rows	Rollout tok.	Update tok.	Harbor time	Use in paper
Main AHE	AHE / AHE-NexAU	DeepSeek-V4-Flash	20	5	20	720	1053.8M	78.1M	14h06m	Main result and diagnostics
Main ACE	ACE / OpenCode	DeepSeek-V4-Flash	20	5	20	720	1115.4M	–	14h33m	Main result
Main TF-GRPO	TF-GRPO / OpenCode	DeepSeek-V4-Flash	20	5	20	1120	1887.9M	3.3M	5h08m	Main result; train uses grouped rollout evidence
GEPA diagnostic	GEPA / OpenCode	DeepSeek-V4-Flash	80	1	1	260	262.0M	–	1h24m	Diagnostic only
AHE batch 10	AHE / AHE-NexAU	DeepSeek-V4-Flash	10	5	40	720	567.8M	125.2M	17h47m	Batch-size ablation
AHE batch 40	AHE / AHE-NexAU	DeepSeek-V4-Flash	40	5	10	720	782.2M	33.6M	10h44m	Batch-size ablation
AHE batch 80	AHE / AHE-NexAU	DeepSeek-V4-Flash	80	5	5	720	769.9M	17.9M	10h09m	Batch-size ablation
AHE batch 80 continue	AHE / AHE-NexAU	DeepSeek-V4-Flash	80	5	5	720	896.2M	16.9M	7h40m	Diagnostic continuation
AHE HLE only	AHE / AHE-NexAU	DeepSeek-V4-Flash	20	5	20	710	625.4M	68.9M	–	Source-diversity ablation; final contaminated
AHE GLM	AHE / AHE-NexAU	GLM-5.1	20	5	20	720	159.8M	111.0M	14h32m	Cross-model diagnostic; final contaminated

Table 9: Saved training-run settings and recorded budget. Task rows include train, update-validation, and final task executions saved by the training run; additional evaluate-only snapshot jobs are reported separately in their own artifacts and are not added here. A dash means the field was not recoverable from normalized records.

C Additional Results

This section provides the supplementary plots referenced by the experiment section. The main paper reports compact epoch-level curves and summary tables; the figures below expose source-group breakdowns, batch-index views, and replay-based fix/forgetting diagnostics. These plots use the same saved task-result records and aggregation conventions as Section 4.

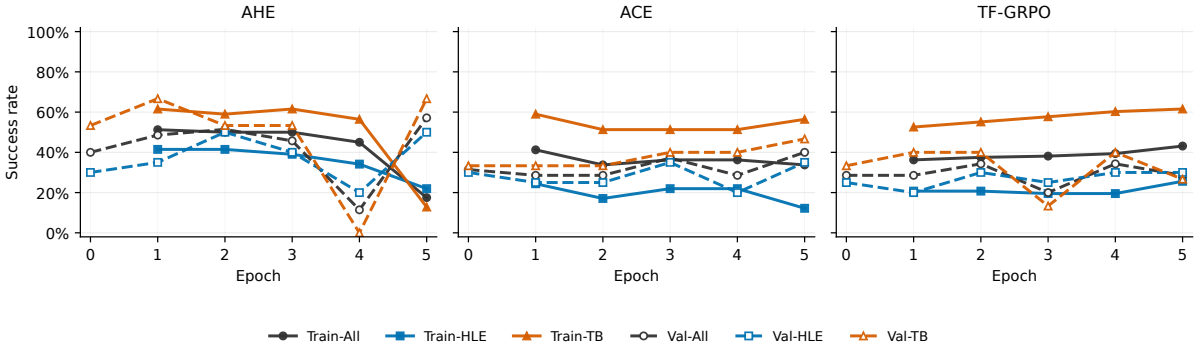

Figure 4: Baseline success-rate breakdown by source group. Each panel reports one method; colors distinguish all tasks, HLE tasks, and Terminal-Bench tasks, while line style distinguishes train and validation curves.

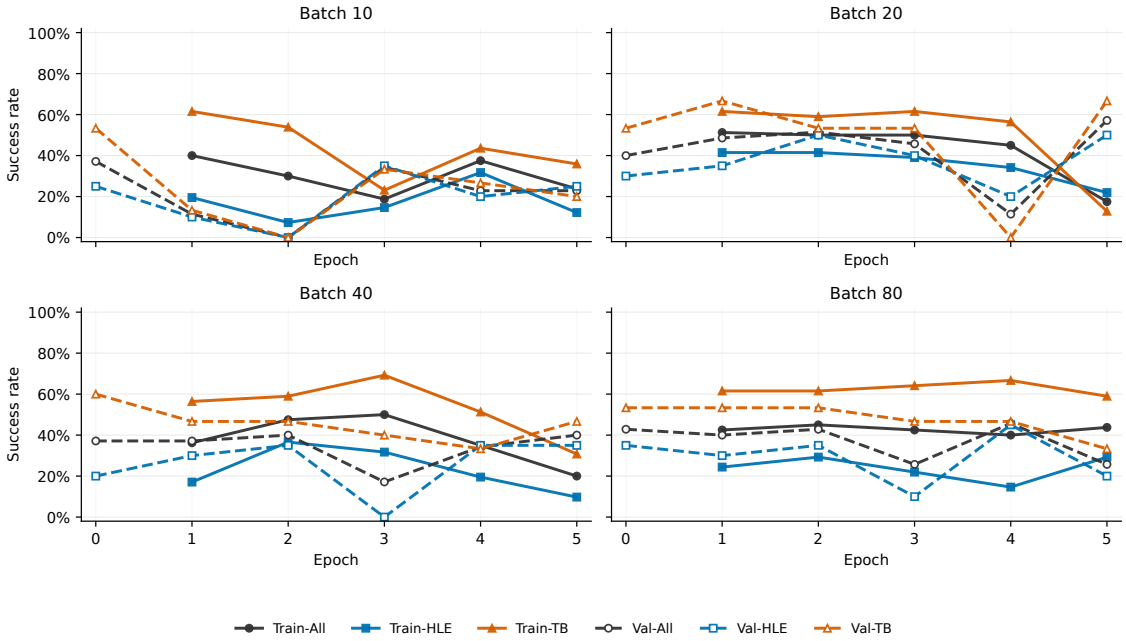

Figure 5: AHE batch-size success-rate breakdown by source group. Each panel reports one batch size; colors distinguish all tasks, HLE tasks, and Terminal-Bench tasks, while line style distinguishes train and validation curves.

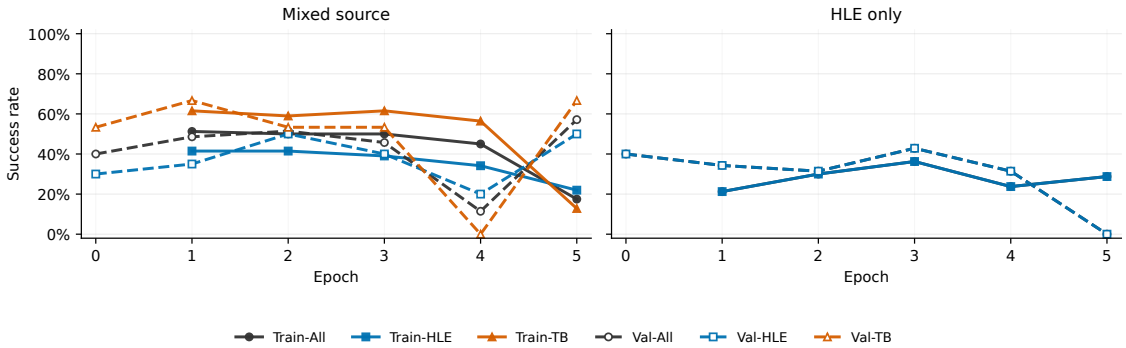

Figure 6: AHE source-diversity success-rate breakdown by source group. Each panel reports one source setting; colors distinguish all tasks, HLE tasks, and Terminal-Bench tasks, while line style distinguishes train and validation curves.

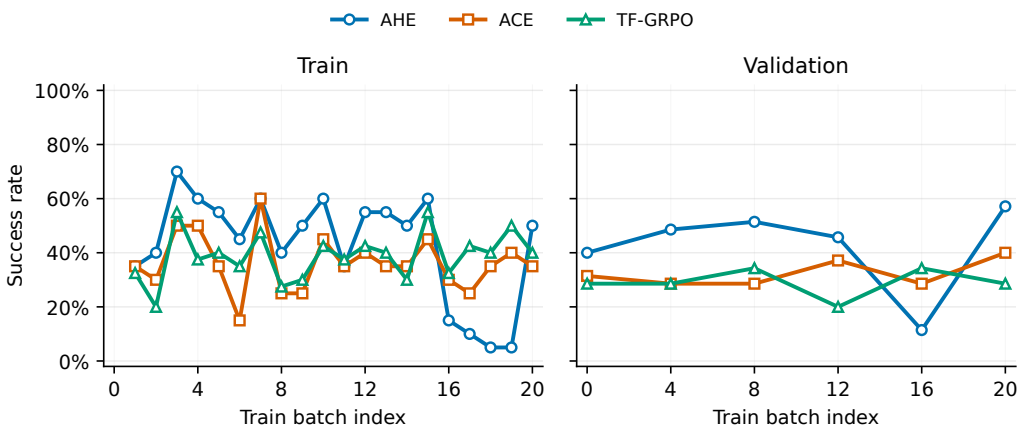

Figure 7: Baseline learning curves by train batch index. Validation points are plotted at the corresponding epoch-end batch index.

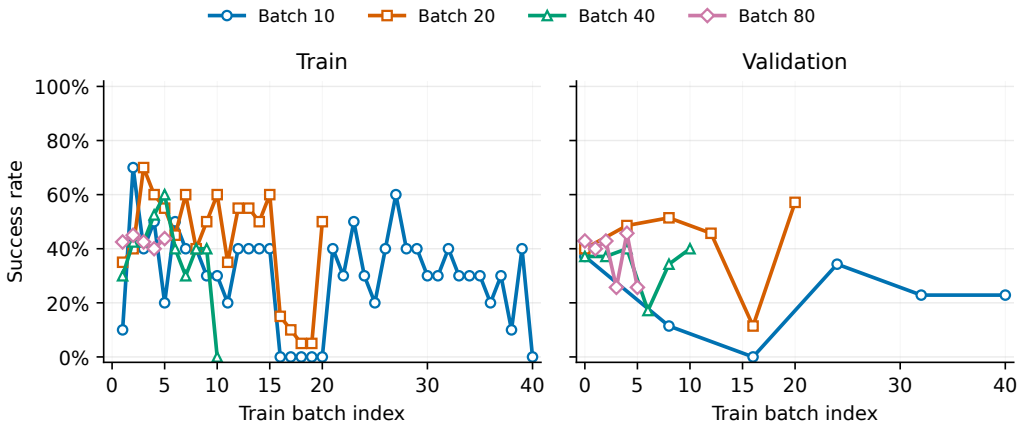

Figure 8: AHE batch-size learning curves by train batch index. Different batch sizes have different numbers of update points; validation points are plotted at the corresponding epoch-end batch index.

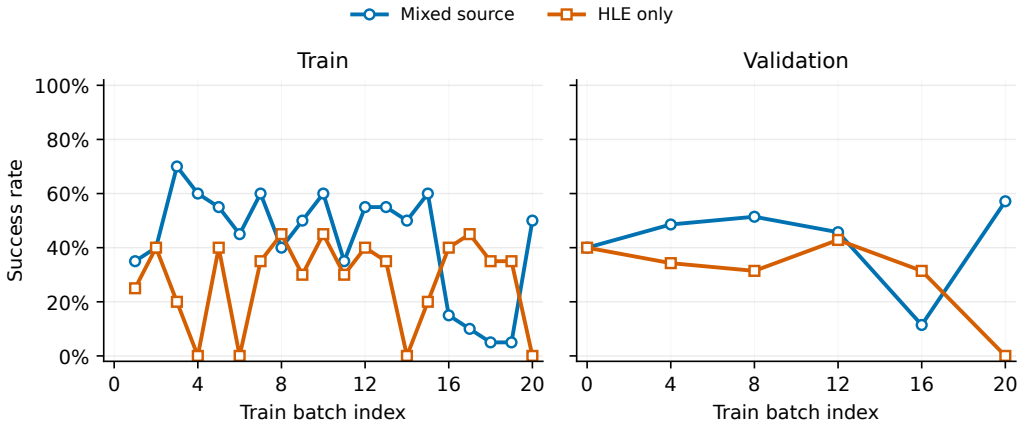

Figure 9: AHE source-diversity learning curves by train batch index. Validation points are plotted at the corresponding epoch-end batch index.

C.1 Cross-Model Continuation Results

The cross-model appendix compares AHE training runs using the same Terminal-Bench + HLE source setting and batch-20 schedule, with DeepSeek-V4-Flash, GLM-5.1, and GPT-5.4 as the training backend. The GPT-5.4 row uses the continuation run from epoch 3 through epoch 5. Final ID and OOD evaluations were not complete for GPT-5.4 at the time of this draft, so Table 10 reports only training-run validation and cost fields.

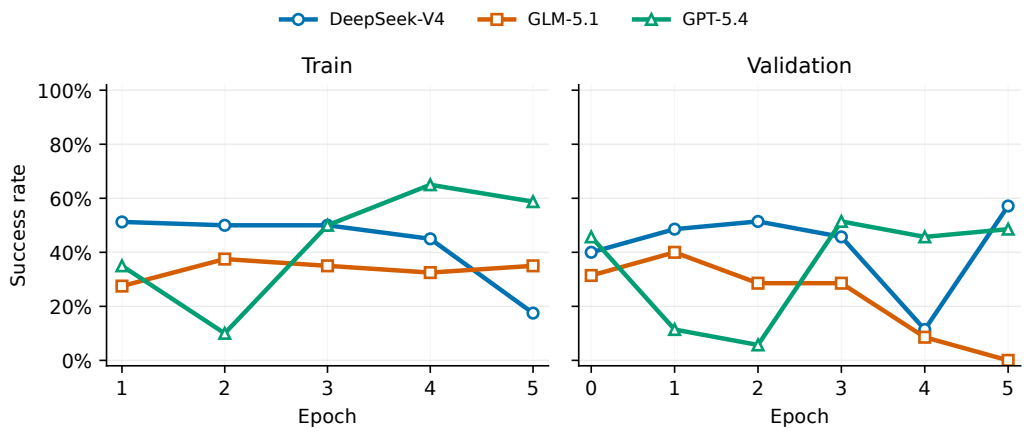

Figure 10: AHE cross-model learning curves. The left panel reports epoch-averaged train success rate, and the right panel reports epoch-end validation success rate.

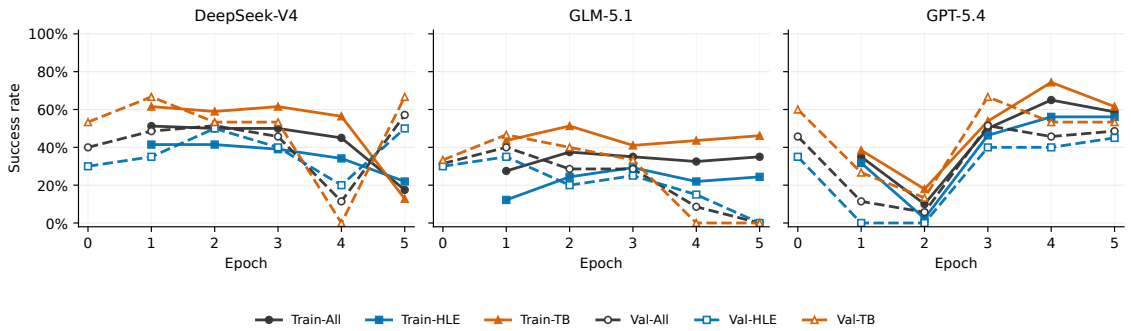

Figure 11: AHE cross-model success-rate breakdown by source group. Each panel reports one training backend; colors distinguish all tasks, HLE tasks, and Terminal-Bench tasks, while line style distinguishes train and validation curves.

Training backend	V_0	V_T	UVG	$V_{\max}$	Best epoch	Rollout tok./task	Update tok./update	Updates	Status
DeepSeek-V4-Flash	40.0	57.1	+17.1	57.1	5	1.46M	3.91M	20/20 changed	complete
GLM-5.1	31.4	0.0	-31.4	40.0	1	0.22M	5.55M	19/20 changed	final validation collapse
GPT-5.4	45.7	48.6	+2.9	51.4	3	0.19M	3.62M	17/20 changed	continuation complete; final ID/OOD pending

Table 10: AHE cross-model continuation summary. Success-rate columns are percentages, and UVG is the final validation gain in percentage points. $V_{\max}$ is the best epoch-end validation score observed during training. Token costs are normalized as rollout tokens per evaluated task/trial and recorded update tokens per SEAGym update call. Final ID and OOD columns are omitted because GPT-5.4 final evaluations were not yet available.

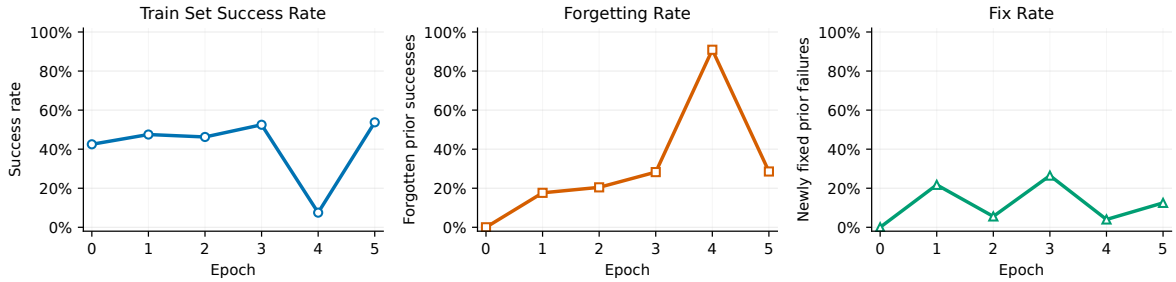

Figure 12: AHE train replay success, forgetting, and fix rate across epochs. Forgetting rate is the fraction of tasks solved by any earlier snapshot that are not solved by the current snapshot; fix rate is the fraction of tasks unsolved by all earlier snapshots that are solved by the current snapshot.

D Integration Details

D.1 Task Index and Visibility

SEAGYM uses a lightweight task index rather than copying benchmark task definitions. Each indexed task stores a stable id, source benchmark reference, task attributes, scoring metadata, and visibility metadata. Executable instructions, environments, verifiers, and raw artifacts remain in the underlying benchmark backend whenever possible.

The agent-visible task view excludes private evaluation metadata, such as reference outputs, private assertions, split membership, and held-out view labels. This separation prevents evaluation metadata from becoming update evidence.

D.2 Rollout and Update Interfaces

The method interface has two roles:

- **rollout adapter**: runs a batch of tasks under the current harness state and returns trajectories, verifier rewards, public errors, and observed cost;
- **baseline adapter**: consumes the trajectory batch, applies the method's native update rule, and saves updated harness state and update summaries.

This split is needed because many benchmark runners instantiate task agents per trial, while self-evolution state must persist outside individual task executions. It also allows different update mechanisms to share the same task schedule and assessment protocol.

D.3 Harbor Backend

The current implementation uses Harbor as the benchmark execution substrate. Harbor runs tasks, environments, verifiers, parallel jobs, and trial artifacts. SEAGYM adds the outer self-evolution schedule, snapshot records, held-out view orchestration, metric inputs, and normalized reports. The integration does not require modifying Harbor source code or redefining Harbor's adapter standard.

E Metric Details

Metric	Definition	Interpretation
Perf	$\frac{1}{\lvert D\rvert}\sum_{x\in D} r(A,x)$	normalized verified task performance
SR	$\frac{1}{\lvert D\rvert}\sum_{x\in D} \mathbb{I}[r(A,x)=1]$	binary success rate
UVG^{prev}	$\text{Perf}(E_i,V)-\text{Perf}(E_{i-1},V)$	local update-validation gain
UVG^{base}	$\text{Perf}(E_i,V)-\text{Perf}(E_0,V)$	cumulative update-validation gain
IDG	$\text{Perf}(A_T,D_I)-\text{Perf}(A_0,D_I)$	final held-out in-domain gain
TG	$\text{Perf}(A_T,D_T)-\text{Perf}(A_0,D_T)$	final cross-domain or held-out transfer
FR	$\max(0,\text{Perf}(A_0,D_R)-\text{Perf}(A_T,D_R))$	replay forgetting or regression
Cost	tokens, tool calls, time, or dollars	efficiency and overhead of evolution

Table 11: Summary of SEAGYM metrics. Primary scores come from verified task outcomes; update-validation and replay metrics explain the trajectory of self-evolution.

E.1 Saved Records

Metrics are computed from saved run records rather than live objects. The run stores evaluation-point summaries, task-level normalized results, verifier outputs, cost records, update summaries, snapshot references, and backend job references. This lets users recompute metrics or change aggregation rules without rerunning task environments.

E.2 Update Labels

Beneficial, neutral, and harmful update labels are diagnostic annotations. They are derived from update-validation deltas, replay deltas, configured thresholds, and optional process judges. They are used for analysis and are not reported as a separate headline metric in the current paper.

E.3 Aggregation

Main tables use domain-level macro averages by default so that domains with more tasks do not dominate the score. Micro averages and task-level breakdowns can be reported in appendix tables for diagnostics.