Test-Time Learning with an Evolving Library

Weijia Xu[1] **Alessandro Sordoni**[1] **Chandan Singh**[1] **Zelalem Gero**[1]

Michel Galley[1] **Xingdi Yuan**[1] **Jianfeng Gao**[1]

[1]Microsoft Research
Correspondence: weijiaxu@microsoft.com

Abstract

We introduce EVOLIB, a test-time learning framework that enables large language models to accumulate, reuse, and evolve knowledge across problem instances without parameter updates or external supervision. Instead of adapting model parameters, our approach maintains a shared library of *knowledge abstractions*, including modular skills and reflective insights, automatically extracted from the model's own inference trajectories. To support continual improvement, we introduce a principled weighting and consolidation mechanism that jointly optimizes for immediate utility and long-term value. This allows simple, instance-specific abstractions to evolve into more general and reusable ones over time. Across challenging benchmarks in mathematical reasoning, code generation, and multi-turn agentic environments, EVOLIB improves substantially over the top test-time scaling and learning methods without ground-truth feedback.[1]

1 Introduction

Humans can learn from past experience and encode that knowledge in abstract, reusable, and extensible forms. This allows them to accumulate and refine knowledge over time, enabling adaptation to novel situations with far less repeated effort. By contrast, large language models (LLMs) typically handle each input problem independently [1–6] or share memory across problem instances by storing and retrieving past problems and trajectories from a global memory [7–9], which hinders LLMs from inducing generalized knowledge or skills across problems.

Test-time learning (TTL) bridges this gap by enabling models to adapt during inference, which improves performance across a sequence of tasks [10–12]. In principle, TTL allows models to correct systematic errors, discover reusable strategies, and build knowledge across problems. However, existing approaches face two fundamental limitations in the context of modern LLMs. First, most methods rely on gradient updates on model parameters to learn effectively from test-time signals. These approaches are not applicable in the increasingly common *black-box* setting, where model weights are inaccessible. Second, many TTL approaches depend on external supervision signals, such as reward functions or example test cases, to guide learning. Such signals are unavailable in many real-world application scenarios.

A complementary line of work, *test-time scaling*, improves performance for black-box LLMs without external supervision, by allocating more computation per instance (e.g. through sampling [1], searching [6], or iterative refinement [4]). While effective, these methods treat each problem in

[1]Code is available at: https://github.com/microsoft/EvoLib.

isolation: the knowledge uncovered during the search process is discarded after inference, preventing the model from inducing generalizable knowledge across problems.

In this paper, we introduce EVOLIB, an evolving library for test-time learning that overcomes these limitations by enabling *knowledge accumulation without parameter updates or external supervision*. The key idea is to maintain a structured, evolving collection of *knowledge abstractions*. The system extracts two types of abstractions from model-generated solutions: (1) *modular skills*, which capture reusable procedures, and (2) *reflective insights*, which encode common errors and corrective strategies.

EVOLIB is governed by two central principles:

- **Abstraction:** Instead of memorizing raw experiences, the library distills them into reusable knowledge units that can be flexibly recomposed across tasks. This enables more efficient transfer compared to memory-based approaches that store unstructured trajectories.
- **Evolution:** The library is continuously updated through iterative abstraction extraction, consolidation and dynamic weight computation. We introduce a novel credit assignment mechanism based on *Information Gain* (IG) and *Future Information Gain* (Future IG), which jointly capture the immediate usefulness of an abstraction and its potential for generating valuable future abstractions. This mechanism promotes the emergence of increasingly useful and extensible abstractions over time.

Importantly, the entire process is *self-supervised* – the model evaluates its own generated solutions and extracts abstractions without relying on external ground-truth signals. This makes the approach applicable to highly challenging tasks and deployment scenarios where external feedback is unavailable or expensive to collect.

We evaluate EVOLIB on a diverse suite of benchmarks spanning mathematical reasoning, code generation, and multi-turn agentic tasks. Across all settings, our method demonstrates consistent improvements over the top test-time scaling and learning approaches with more efficient token usage. Further results in the continual learning setting support that EVOLIB enables continual learning that is less dependent on task ordering, whereas existing TTL methods based on linear knowledge updates are more sensitive to the ordering.

2 Related Work

Test-time scaling and adaptation. There is a growing body of work that improves LLM performance by allocating additional computation at inference time rather than updating model parameters. Representative approaches include best-of-N sampling and self-consistency [1], self-verification with and without feedback [2–4], iterative refinement and aggregation [5, 6]. While effective, these methods treat each problem independently: any knowledge discovered during inference is discarded after solving a single instance. By contrast, EVOLIB explicitly retains and reuses knowledge across problems, enabling continual improvement over a stream of tasks.

More closely related are *test-time learning* and *test-time training* methods that adapt model behavior at test time. Recent work formulates test-time learning for LLMs as self-supervised optimization on unlabeled test data [10] or few-shot learning on example test cases [11], often using lightweight parameter updates [13, 14]. These methods require gradient updates on model parameters, whereas EVOLIB performs test-time learning purely through abstraction induction and evolution, without any parameter modification.

Memory-augmented LLMs. Several recent works augment LLMs with external memory to persist information across queries or interactions. Retrieval-augmented generation and dynamic prompt construction store and retrieve raw past experience [7, 8]. [9] introduces ExpRAG, a simple framework for retrieving and reusing prior experience, along with a variant that allows the agent to iteratively refine the retrieved memory. While these approaches demonstrate the importance of shared memory across queries, they primarily store unstructured trajectories or episodic experiences. EVOLIB differs by distilling experience into *extensible abstractions* and by explicitly promoting extension and consolidation to produce more general knowledge units.

Skill and knowledge discovery. The idea of learning reusable skills from experience has a long history in program synthesis and cognitive science [15–17]. One line of work learns how to use

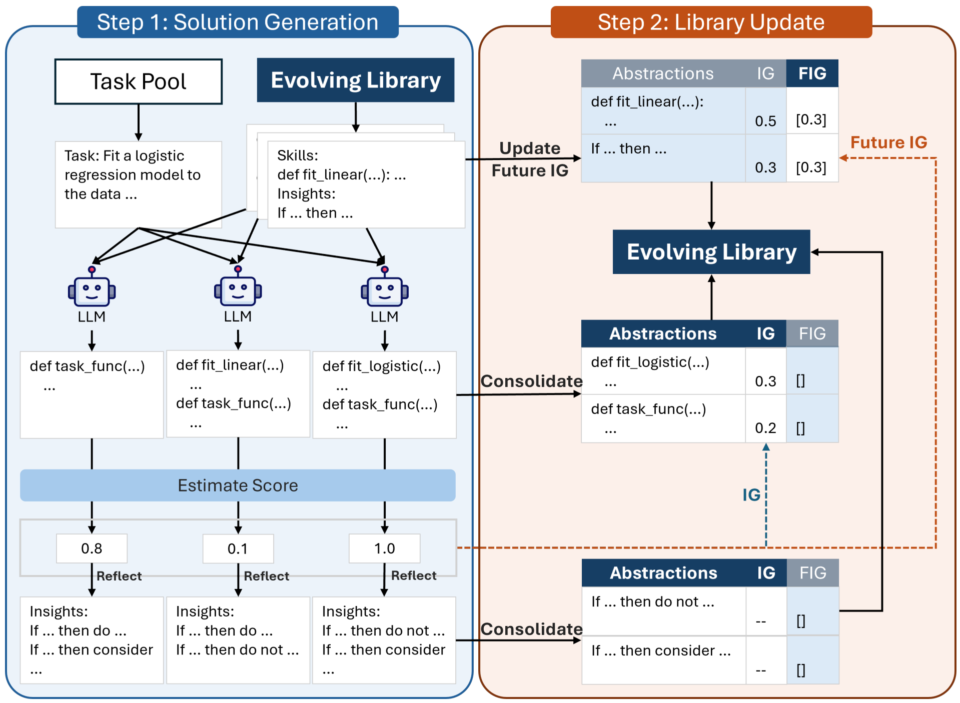


Figure 1: Overview of the EvoLib algorithm. EvoLib performs test-time learning by repeatedly: (i) solving tasks using sampled abstractions, (ii) extracting new abstractions, consolidating them into the library, and propagating credit to both new and previously used abstractions via Information Gain (IG) and Future IG.

existing tools effectively [18, 19]. Another line of work focuses on extracting reusable skills, such as subroutines or functions, from solved programs or successful trajectories [20–24], or inducing reflective insights from model-generated programs or trajectories together with feedback [25–29]. Furthermore, inspired by the recent development of agentic "autoresearch" loops [30–32], recent works propose skill evolution by iteratively refining a skill set based on new trajectories [33, 34]. Most existing approaches learn such libraries on training tasks where external success signals are available, whereas our approach can be applied solely at test-time.

Closest to our setting, AWM [35] proposes to maintain a memory of reusable routines extracted from action trajectories with optional external feedback, while Dynamic Cheatsheet [36] induces high-level principles and procedural knowledge via self-reflection. Similarly, a concurrent work constructs a memory of reasoning strategies distilled from self-judged successful and failed experiences [37]. However, these approaches either treat memory as a monolithic artifact that is refined holistically [36] or append new items into the memory in a linear fashion [35, 37], making them susceptible to three major failure modes: *misgrouping*, where experiences from different types of tasks are consolidated into the same abstraction; *interference*, where repeated LLM rewriting strips away applicability conditions and turns useful lessons into overly broad guidance; and *overfit*, where memory becomes tied to narrow surface patterns rather than reusable strategies [38]. EvoLib addresses these issues by introducing a principled weighting, sampling, and consolidation mechanism that drives the emergence and nonlinear evolution of abstractions, guided by immediate and long-term utility across instances (see the extended results in Section B).

3 EvoLib

EvoLib is characterized by two key properties. First, **abstraction** – the library stores generalizable knowledge distilled from raw trajectories, making it easier for LLMs to adapt and reuse past experience across problems compared to memory-based methods that store raw trajectories. Second, **evolution** – the library is continuously updated through a consolidation mechanism and a dynamic weighting scheme based on Information Gain and Future Information Gain, enabling abstractions to improve over time.

3.1 Problem Formulation

We represent an agent augmented with EvoLib as a tuple $(\Phi, \mathcal{K})$, where Φ is the base LLM and $\mathcal{K}$ is the library. Starting with an empty library $\mathcal{K}_0 = \emptyset$, the agent is given a sequence of tasks $\{x_1, x_2, ..., x_T\}$, each sampled from the task pool $\mathcal{T}$, and the library $\mathcal{K}_t$ evolves with the history. At step t, given the input problem x_t and the current library $\mathcal{K}_{t-1}$, the agent samples abstractions from the library and generates a solution $\tilde{y}_t$ for x_t:

$$\{z_{t-1}\} \sim \mathrm{Sample}(\mathcal{K}_{t-1}, x_t),$$
$$\tilde{y}_t \sim P_\Phi(\cdot \mid x_t, \{z_{t-1}\}),$$

then extracts new abstractions $\{z_t\}$ from $\tilde{y}_t$:

$$\{z_t\} = \mathrm{Extract(x_t, \tilde{y}_t)}.$$

Finally, the agent updates the library based on the current library and this attempt:

$$\mathcal{K}_t = \mathrm{Update\_Library}[\mathcal{K}_{t-1}, (x_t, \tilde{y}_t, \{z_{t-1}\}, \{z_t\})].$$

3.2 Library Representation

The library $\mathcal{K}$ is represented as a weighted collection of knowledge abstractions. It includes two types of knowledge: 1) *modular skills* that can be adapted to solve new tasks. For coding tasks, these are functions; for reasoning tasks, they are sub-problems with corresponding solutions extracted from complete solution trajectories; for agentic tasks, they are sub-tasks with corresponding workflows extracted from complete action trajectories. And 2) *reflective insights* in natural language capturing lessons learned from past trajectories, including common mistakes and corrective strategies (e.g. "If the task involves logistic regression, then do not forget to scale the features").

Sampling. When sampling from the library to solve a new task, we first filter the library entries based on their embedding-based similarity to the task description and then use weighted sampling among the filtered entries based on their weights $w(z \mid \mathcal{K})$. These weights are updated dynamically after generating new solutions for a problem based on the Information Gain and Future Information Gain described in Section 3.2.2.

3.2.1 Extracting Abstractions

We extract *modular skills* $\{z_t\}$ directly from model outputs $\tilde{y}_t$, as these skills are typically *inherent* in the model's reasoning steps either explicitly or implicitly. For coding tasks, we extract functions as skills directly from the synthesized programs. For reasoning and agentic tasks, we extract skills by breaking down the reasoning steps or an action-observation sequence into several sub-modules and summarizing each sub-module into a new, self-contained skill through an LLM call.

To extract *reflective insights* $\{z_t\}$, we first use the LLM itself to estimate the score $S_\Phi(\tilde{y}_t \mid x_t) \in [0, 1]$ for the model solution $\tilde{y}_t$ (with optional textual feedback) [39, 2]. For coding tasks, $S_\Phi(\tilde{y}_t \mid x_t)$ is obtained by instructing the LLM to generate synthetic test cases for the problem x_t and measuring the pass rate of the solution program $\tilde{y}_t$ against the synthetic test. For reasoning tasks, $S_\Phi(\tilde{y}_t \mid x_t)$ is obtained by majority voting [1] and LLM-as-a-Judge [39] to break tie. For multi-turn agentic tasks, the score is obtained by passing the action-observation sequence $\tilde{y}_t$ to the LLM and asking it to judge how many sub-goals in the task x_t have been accomplished. Next, we extract the insights by instructing the LLM to reflect on the solution $\tilde{y}_t$ based on x_t and $S_\Phi(\tilde{y}_t \mid x_t)$ and summarize it into several self-contained insights.

3.2.2 Weighting Mechanism

The core mechanism that enables evolution is credit propagation along the evolution chains of abstractions: useful abstractions not only receive direct reward (through *Information Gain*), but also

Algorithm 1 EVOLIB

Input: Task pool $\mathcal{T}$ of size N, initial library $\mathcal{K}_0 = \emptyset$, model Φ, number of iterations T
Initialize best solutions $\{y_j^*\}_{j=1}^N$ for each task $x_j \in \mathcal{T}$
for t from 1 to T **do**
  Draw a task $x_t \sim \mathcal{T}$ # Solution Generation
  for k from 1 to K **do**
    Sample M abstractions $\{z_{t-1}^{(k)}\} \sim \mathrm{Sample}(\mathcal{K}_{t-1}, x_t)$
    Sample solution trajectory $\tilde{y}_t^{(k)} \sim P_\Phi(\cdot \mid x_t, \{z_{t-1}^{(k)}\})$
    Estimate the accuracy score $S_\Phi(\tilde{y}_t^{(k)} \mid x_t)$
    if $S_\Phi(\tilde{y}_t^{(k)} \mid x_t) > S_\Phi(y_t^* \mid x_t)$ **then**
      $y_t^* \leftarrow \tilde{y}_t^{(k)}$
    end if
  end for
  $k^* \leftarrow \arg\max_k S_\Phi(\tilde{y}_t^{(k)} \mid x_t)$
  $\tilde{y}_t \leftarrow \tilde{y}_t^{(k^*)}$

  $\mathcal{K}_{\mathrm{new}} \leftarrow \mathcal{K}_{t-1}$ # Library Update
  Extract new abstractions $\{z_t\} \leftarrow \mathrm{Extract}(x_t, \tilde{y}_t)$
  for z_t in $\{z_t\}$ **do**
    Compute $\mathrm{IG}(z_t \mid x_t, \mathcal{K}_{t-1})$ based on Eq. (3) if z_t is a skill else 0
    $\mathcal{K}_{\mathrm{new}} \leftarrow \mathrm{Consolidate}[(z_t, \mathrm{IG}(z_t \mid x_t, \mathcal{K}_{t-1})), \mathcal{K}_{\mathrm{new}}]$
  end for
  for z_{t-1} in $\{z_{t-1}^{(k)}\}$ **do**
    Compute $\mathrm{FutureIG}(z_{t-1} \mid x_t, \mathcal{K}_{t-1})$ based on Eq. (6)
    $\mathcal{K}_{\mathrm{new}} \leftarrow \mathrm{Update\_FutureIG}[(z_{t-1}, \mathrm{FutureIG}(z_{t-1} \mid x_t, \mathcal{K}_{t-1})), \mathcal{K}_{\mathrm{new}}]$
  end for
  $\mathcal{K}_t \leftarrow \mathcal{K}_{\mathrm{new}}$
end for
Output: Final library $\mathcal{K}_T$, best solutions $\{y_j^*\}_{j=1}^N$

propagate credit backward to the abstractions that enabled their creation (through *Future Information Gain*). Specifically, we focus on the evolution chains:

$$\{z_1\} \xrightarrow{x_2, \tilde{y}_2} \{z_2\} \xrightarrow{x_3, \tilde{y}_3} \ldots \xrightarrow{x_{t-1}, \tilde{y}_{t-1}} \{z_{t-1}\} \xrightarrow{x_t, \tilde{y}_t} \{z_t\} \xrightarrow{x_{t+1}, \tilde{y}_{t+1}} \ldots,$$

where each step of the evolution $\{z_{t-1}\} \xrightarrow{x_t, \tilde{y}_t} \{z_t\}$ is realized by:

$$\tilde{y}_t \sim P_\Phi(\cdot \mid x_t, \{z_{t-1}\}),\ \{z_t\} = \mathrm{Extract}(x_t, \tilde{y}_t).$$

If $\{z_t\}$ is proven to be useful for producing a high-quality solution for the task x_t, the credit should be propagated not only to the new abstractions $\{z_t\}$, but also to the preceding abstractions $\{z_{t-1}\}, \{z_{t-2}\}, \ldots, \{z_1\}$ along the evolution chain. In practice, to balance between efficacy and computational cost, we propagate the credit by two steps, i.e. to $\{z_t\}$ and $\{z_{t-1}\}$.

More formally, to propagate the credit to $\{z_t\}$, we first define the baseline score for x_t given $\mathcal{K}_{t-1}$:

$$\mu_{\mathrm{base}}(Y \mid x_t, \mathcal{K}_{t-1}) = \mathbb{E}_{\tilde{y}_t, Z_{t-1}}\left[S_\Phi(\tilde{y}_t \mid x_t)\right], \tag{1}$$

where $\tilde{y}_t$ and Z_{t-1} follow $\tilde{y}_t \sim P_\Phi(\cdot \mid x_t, Z_{t-1})$ and $Z_{t-1} \sim \mathrm{Sample}(\mathcal{K}_{t-1}, x_t)$ in the expectation.

We then define the conditional score given that z_t is present in the extracted abstractions:

$$\mu_{\mathrm{cond}}(Y \mid x_t, \mathcal{K}_{t-1}, z_t) = \mathbb{E}_{\tilde{y}_t, Z_{t-1}}\left[S_\Phi(\tilde{y}_t \mid x_t) \mid z_t \in \mathrm{Extract}(x_t, \tilde{y}_t)\right]. \tag{2}$$

Finally, we define **Information Gain (IG)**, which measures how much better the model Φ can solve x_t when z_t is present in the extracted abstractions compared to the average baseline:

$$\mathrm{IG}(z_t \mid x_t, \mathcal{K}_{t-1}) = \log\big(\mu_{\mathrm{cond}}(Y \mid x_t, \mathcal{K}_{t-1}, z_t)\big) - \log\big(\mu_{\mathrm{base}}(Y \mid x_t, \mathcal{K}_{t-1})\big). \tag{3}$$

Furthermore, to propagate the credit back to $\{z_{t-1}\}$, we first define the baseline score where z_{t-1} is not present in the previously sampled abstractions:[2]

$$\mu_{\text{base}}(Y \mid x_t, \mathcal{K}_{t-1}, \overline{z_{t-1}}) = \mathbb{E}_{\tilde{y}_t, Z_{t-1}} \left[S_\Phi(\tilde{y}_t \mid x_t) \mid z_{t-1} \notin Z_{t-1} \right], \tag{4}$$

and the conditional score where z_{t-1} is present:

$$\mu_{\text{cond}}(Y \mid x_t, \mathcal{K}_{t-1}, z_{t-1}) = \mathbb{E}_{\tilde{y}_t, Z_{t-1}} \left[S_\Phi(\tilde{y}_t \mid x_t) \mid z_{t-1} \in Z_{t-1} \right]. \tag{5}$$

And finally, we define **Future IG**, which quantifies how much better the model can solve x_t given z_{t-1} as one of the preceding abstractions, compared to the baseline score:

$$\text{FutureIG}(z_{t-1} \mid x_t, \mathcal{K}_{t-1}) = \log\big(\mu_{\text{cond}}(Y \mid x_t, \mathcal{K}_{t-1}, z_{t-1})\big) - \log\big(\mu_{\text{base}}(Y \mid x_t, \mathcal{K}_{t-1}, \overline{z_{t-1}})\big). \tag{6}$$

Future IG can be interpreted as estimating the contribution of an abstraction to the generation of useful future abstractions.

Overall, we compute the weight of each abstraction z in the library based on its IG and Future IG scores:

$$w(z \mid \mathcal{K}) = \tau \max_{x \in \mathcal{T}} \text{IG}(z \mid x, \mathcal{K}) + \mathbb{E}_{x \sim \mathcal{T}} \text{FutureIG}(z \mid x, \mathcal{K}), \tag{7}$$

where τ is a hyperparameter controlling the relative importance of immediate versus future gain. In practice, we use $\tau = 1$ for all skills and $\tau = 0$ for all insights, as skills can directly contribute to solution construction, whereas insights may not. Future IG is computed for all abstractions (including skills and insights), as both can influence future solution and abstraction generation.

We use the maximum IG over tasks to capture the peak utility of an abstraction, encouraging any abstraction with immediate utility to be sampled and evolve in the future. For Future IG, we take the expectation of Future IG over tasks randomly sampled from the task pool by maintaining a list of Future IG scores for each z, initialized by an empty list. At each iteration, when z_{t-1} is sampled from the library and integrated in the context to solve a task x_t, we compute z_{t-1}'s Future IG given x_t based on Eq. (6) and add it to its Future IG list. When computing $w(z \mid \mathcal{K})$, we estimate the expected Future IG by averaging over the Future IG scores in the list.

Updating Future IG dynamically in this manner makes the agent less prone to the noise in self-evaluated scores, as we take the average over Future IG scores across iterations. It also enables continual evolution of the library – when a more advanced abstraction covering a broader range of use cases emerges, it typically receives high Future IG, while older, less general abstractions exceeded by the new abstraction will receive decreasing Future IG. As a result, more advanced abstractions gradually become more prevalent with higher sampling probability, while older ones are progressively de-emphasized without explicit removal.

3.3 Library Update

Building on the abstraction extraction (Section 3.2.1) and weighting mechanism (Section 3.2.2), we now describe how the library is updated. For each abstraction in the library, we maintain its IG score and a list of Future IG scores obtained on previous tasks. The library is updated in two ways: 1) **abstraction consolidation** where new abstractions are consolidated with similar entries and added to the library, and 2) **updating Future IG** where for each abstraction integrated in the context for solving the current task, we update its Future IG list with the new Future IG score.

Abstraction consolidation. When a new abstraction is extracted from a trajectory, we first retrieve the most similar abstraction of the same type from the library using embedding-based similarity. We then prompt the LLM to consolidate the new abstraction into the most similar one from the library when the abstractions are semantically and functionally similar. If the two abstractions can be merged, the IG and Future IG scores of the old item are also merged into the new consolidated entry. Otherwise, the new abstraction is added as a separate entry (with the current IG score and an empty Future IG list) into the library. This consolidation step is critical for promoting the emergence of abstractions that generalize beyond specific problems.

[2] Note that IG uses the unconditional baseline, while Future IG uses an exclusion baseline to avoid bias introduced by repeated sampling of the same abstraction.

Algorithm 1 summarizes the full procedure, alternating between solution generation and library update (as illustrated in Fig. 1). In the solution generation step, the agent samples a set of abstractions from the current library to construct a contextual prompt, generates and estimates the scores of candidate solutions. In the library update step, newly extracted abstractions are consolidated into the library, and the Future IG scores of the sampled abstractions are updated. The agent iterates through the two steps across successive tasks, enabling the library to continuously grow and evolve.

4 Experiments

4.1 Experimental Setup

Evaluation datasets. We evaluate EvoLib on a diverse set of benchmarks, including mathematical reasoning, code generation, and multi-turn agentic tasks, in two different settings: 1) static benchmarks, where the agent learns through multiple iterations over the whole dataset, and 2) continual learning, where tasks are presented in a stream and each task is attempted only once. In both settings, the agent learns in a self-supervised manner without access to the ground-truth answers or tests.

For the test-time learning setting, we evaluate it on challenging math and coding benchmarks including a) exam questions from Harvard–MIT Mathematics Tournament (**HMMT**) 2025–2026, which is an elite high-school competition featuring complex mathematical reasoning, b) **BigCodeBench Hard** [40], which contains 148 challenging programming tasks designed to evaluate compositional reasoning and the use of existing function calls for real-world programming, and c) **LiveCodeBench v6 Hard** [41], which contains 80 highly challenging competitive coding problems.

For the continual learning setting, we evaluate it on multi-turn agentic tasks, including the **ScienceWorld** [42] and **PDDL** [43] tasks from the AgentBoard benchmark suite [44]. These tasks require multi-round interaction with a partially observable environment. We adopt the task order in the original dataset in our evaluation. See more dataset details in Section A.1.

LLMs. We evaluate the efficacy of EvoLib using both general-purpose LLMs, such as GPT-4o (on BigCodeBench), and LLMs featuring effective reasoning and coding, such as o4-mini (on HMMT, LiveCodeBench, ScienceWorld, and PDDL). On tasks that require intensive reasoning, such as HMMT and LiveCodeBench, we use o4-mini with high reasoning effort, while on multi-turn agentic tasks, we use o4-mini with low reasoning effort.

Baselines. We compare EvoLib against both test-time scaling (TTS) and test-time learning (TTL) approaches. For TTS, we adopt a) **Best-of-N**, which generates N independent solutions and selects the one with the highest estimated accuracy, and b) **Recursive Self-Aggregation (RSA)** [6], which iteratively refines and improves a population of candidate solutions through aggregation of subsets.[3] For TTL, we include **ExpRAG** [9], an agent augmented with a memory of past trajectories and **Dynamic Cheatsheet (DC)** [36], a strong TTL framework that augments an LLM with an evolving textual memory of procedural knowledge.

Evaluation metrics. On HMMT, we evaluate the accuracy of the LLM's solutions against the ground-truth answers using the grading script in [45]. On BigCodeBench and LiveCodeBench, we measure task performance by the pass rate on the true test cases for each problem. On ScienceWorld and PDDL, we adopt the success rate and progress rate from the AgentBoard [44]. On each benchmark, we compare the performance of various methods under the same cost budget measured by weighted token count, where the input token count is weighted by $\lambda = 1$ while the output token count is weighted by $\lambda = 4$.[4]

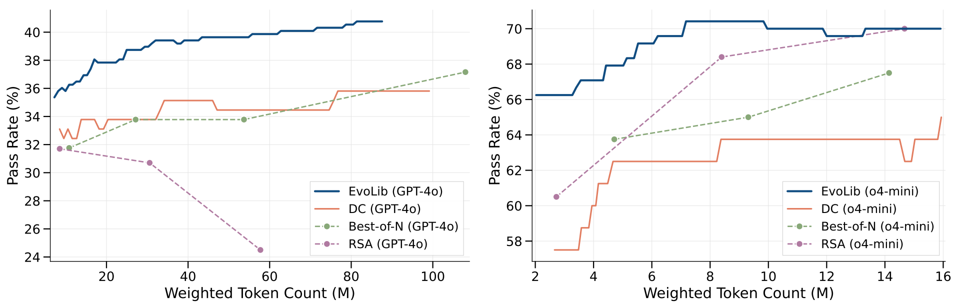
4.2 Main Result: EvoLib Consistently Outperforms Baselines

Comparison on static benchmarks. Table 1 shows that EvoLib achieves the strongest average performance across all evaluated settings, including math, code generation, and multi-turn agentic tasks. In the static setting, EvoLib improves over base sampling by 11-20%, while achieving better or competitive performance compared with the best TTS baseline. On BigCodeBench, GPT-4o

[3] We exclude the TTS baselines on the agentic tasks, as each task is attempted only once through multiple rounds.
[4] The weights are based on the typical per-token rates of input and output tokens for proprietary LLMs.

Table 1: Performance comparing EvoLib against base sampling, test-time scaling (TTS) methods, including Best-of-N and Recursive Self-Aggregation (*RSA*), along with test-time learning (TTL) baselines, including ExpRAG and Dynamic Cheatsheet (*DC*), across math (*HMMT*), coding (*BigCodeBench* and *LiveCodeBench*), and multi-turn agentic tasks (*ScienceWorld* and *PDDL*). For each task, we report the average score over three runs for each method. A dash (–) indicates the method is not applicable to that evaluation setting. An asterisk (*) denotes that the best score(s) is significantly higher than the second-best on each benchmark based on paired student's t-test with $p < 0.05$.

Method	HMMT	BigCodeBench	LiveCodeBench	ScienceWorld		PDDL	
	Accuracy	Pass Rate	Pass Rate	Success	Progress	Success	Progress
Base	57.0	29.7	58.8	53.3	82.9	62.8	79.1
Best-of-N	74.2	37.2	67.5	–	–	–	–
RSA [6]	67.2	31.7	**70.0***	–	–	–	–
ExpRAG [9]	60.2	25.0	51.3	51.1	80.4	39.4	56.7
DC [36]	66.7	35.8	65.0	55.9	83.6	65.0	83.6
EvoLib	**77.4***	**40.8***	**70.0***	**57.4**	**84.3**	**72.8***	**86.2**



(a) BigCodeBench

(b) LiveCodeBench

Figure 2: Cost–performance curves comparing EvoLib with competitive baselines on (a) BigCodeBench and (b) LiveCodeBench. Each curve plots performance (y-axis) as the test-time compute cost (x-axis) increases for each method.

augmented with EvoLib even surpasses the strongest reported LLMs (e.g., *Gemini-Exp-1206* at 40.5%) on the leaderboard. This suggests that test-time learning can substantially close, and in some cases overcome, the gap between less and more powerful base models.

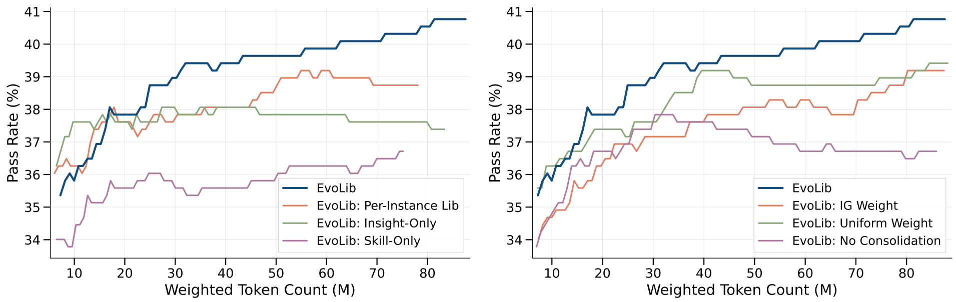
Furthermore, as shown in the cost-performance curves in Figs. 2 and B.1, EvoLib is more token-efficient than both TTS baselines – it yields larger performance gains under tighter cost budgets (+6-16% on BigCodeBench and +3-6% on LiveCodeBench when the budget is reduced by half). These gains indicate that learning and sharing knowledge across problem instances yields consistent improvements over TTS.

Compared to TTL baselines, EvoLib consistently outperforms DC, the best TTL baseline, by 5-10% across three benchmarks. As shown in Fig. 2, it is also more token-efficient than DC.

Comparison in continual learning setting. Table 1 also highlights that EvoLib outperforms all existing TTL methods in the continual learning setting, improving over both ExpRAG and DC baselines. These results demonstrate that EvoLib transfers what it has learned on past tasks to future tasks more effectively than the existing TTL methods.

4.3 Ablation Study

Both abstraction types and cross-instance sharing matter. Fig. 3(a) ablates the types of abstractions stored in the library and how the library is shared. Using only modular skills or only reflective insights underperforms the full variant, which suggests that the two abstraction types contribute complementary value: insights capture higher-level corrective signals that guide future



(a) Ablating Abstraction Types (b) Ablating Weighting and Consolidation

Figure 3: Ablation study on BigCodeBench. (a) Comparison of variants using different abstraction types or per-instance libraries. (b) Comparison of variants with different weighting strategies and without consolidation. Each plot shows the performance in pass rate (y-axis) as a function of test-time compute cost (x-axis).

attempts (especially useful at the early stage), while skills provide solution components that can be reused and recombined to solve more complex tasks. In addition, the per-instance variant lags behind the shared library, especially at larger budgets, which provides strong evidence that the library generalizes beyond problem-specific details. It also supports the claim that cross-problem knowledge reuse and evolution, rather than iterative knowledge refinement within each problem, drives further performance gain.

Abstraction consolidation leads to substantial gain Fig. 3(b) shows the impact of abstraction consolidation. Disabling consolidation hurts performance, particularly as compute increases. We further analyze the role of abstraction consolidation in Section B. Fig. B.2 and Table B.1 indicate that the consolidation step helps control memory growth and turns task-specific abstractions into more generic ones that apply across multiple problems.

Weighting and consolidation are both needed for sustained improvement. Fig. 3(b) also shows the impact of IG and Future IG scores in the weighting mechanism. Uniform weighting leads to weaker scaling at higher compute budgets, where the library grows larger and effective sampling becomes increasingly important. Removing the Future IG score also degrades performance substantially, as it is crucial for selecting abstractions that can evolve into more advanced ones.

4.4 Robustness To Task Ordering

We further study robustness to task ordering in the continual learning settings on PDDL, which contains multiple distinct task types (see Table B.2 and discussion in Section B for details). We find that EvoLib consistently outperforms DC across easy-to-hard, hard-to-easy, and random task streams, with the largest gain (+9%) under random ordering. The result indicates that EvoLib can continually learn from diverse tasks even when they are interleaved, suggesting its practical advantage in real-world scenarios where an agent must handle and learn from a mixed stream of heterogeneous user requests without relying on a structured curriculum.

5 Discussion

This paper introduces EvoLib, a framework that enables black-box LLMs to self-improve at test time through abstraction induction, reuse, and evolution. A key property of EvoLib is its favorable scaling behavior: performance improves consistently as more problems are solved and more compute is invested, without external supervision. This draws a direct analogy to human cognitive development. Just as humans build progressively more abstract mental models through experience, the library evolves from concrete, task-specific functions toward more general, reusable abstractions. Experiments on mathematical reasoning, code generation, and interactive agentic tasks show that EvoLib consistently outperforms existing test-time scaling and test-time learning methods with lower token cost. The framework offers a promising direction for LLMs to learn and self-improve during deployment without requiring gradient-based fine-tuning or supervision signals.

Limitations. The current EvoLib framework assumes that the model's own judgment of a solution provides more useful signals than noise for self-evaluation. Thus, the method works best for tasks in which verifying a solution is easier than generating one (e.g., coding tasks with executable test cases). For tasks where verification requires comparable or even greater effort than solution generation, a powerful external verifier model may be necessary to reliably assess each candidate solution. Additionally, the current evaluation focuses on math, coding, and agentic tasks. It is unclear how well the approach generalizes to other domains, such as open-ended question-answering or scientific tasks, with different knowledge types and feedback characteristics.

Future work. One promising direction is to enable LLMs to meta-learn the design of the library itself, including the choice of abstraction forms, the weighting mechanism, and the library update algorithm. Rather than fixing these components, the system could optimize them over time based on previous tasks and sampled trajectories, the structure of the task stream, potentially leading to more efficient test-time learning schemes. Another direction is to deploy the framework in long-horizon settings with a mixed stream of diverse tasks. Such settings would better reflect real-world usage and help identify challenges related to scalability, distribution shift, and knowledge retention.

References

[1] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[2] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore, December 2023. Association for Computational Linguistics.

[3] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[4] Yufan Zhuang, Chandan Singh, Liyuan Liu, Yelong Shen, Dinghuai Zhang, Jingbo Shang, Jianfeng Gao, and Weizhu Chen. Test-time recursive thinking: Self-improvement without external feedback. *arXiv preprint arXiv:2602.03094*, 2026.

[5] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics.

[6] Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, et al. Recursive self-aggregation unlocks deep thinking in large language models. *arXiv preprint arXiv:2509.26626*, 2025.

[7] Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. Memory-assisted prompt editing to improve GPT-3 after deployment. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2833–2861, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[8] Tao Feng, Pengrui Han, Guanyu Lin, Ge Liu, and Jiaxuan You. Thought-retriever: Don't just retrieve raw data, retrieve thoughts, 2024.

[9] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

[10] Jinwu Hu, Zitian Zhang, Guohao Chen, Xutao Wen, Chao Shuai, Wei Luo, Bin Xiao, Yuanqing Li, and Mingkui Tan. Test-time learning for large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 24823–24849. PMLR, 13–19 Jul 2025.

[11] Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 942–963. PMLR, 13–19 Jul 2025.

[12] Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

[13] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

[14] Yuri Kuratov, Matvey Kairov, Aydar Bulatov, Ivan Rodkin, and Mikhail Burtsev. Gradmem: Learning to write context into memory with test-time gradient descent. *arXiv preprint arXiv:2603.13875*, 2026.

[15] Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

[16] Kurt VanLehn. Cognitive skill acquisition. *Annual review of psychology*, 47(1):513–539, 1996.

[17] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[18] Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, et al. Walt: Web agents that learn tools. *arXiv preprint arXiv:2510.01524*, 2025.

[19] Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

[20] Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. ReGAL: Refactoring programs to discover generalizable abstractions. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 46605–46624. PMLR, 21–27 Jul 2024.

[21] Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*, 2025.

[22] Yimeng Liu, Misha Sra, Jeevana Priya Inala, and Chenglong Wang. Reuseit: Synthesizing reusable ai agent workflows for web automation. In *Proceedings of the 31st International Conference on Intelligent User Interfaces*, pages 885–908, 2026.

[23] Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: From trajectories to reusable expertise for continual llm agent refinement. *arXiv preprint arXiv:2601.22758*, 2026.

[24] Matthew Ho, Chen Si, Zhaoxiang Feng, Fangxu Yu, Yichi Yang, Zhijian Liu, Zhiting Hu, and Lianhui Qin. Arcmemo: Abstract reasoning composition with lifelong llm memory. *arXiv preprint arXiv:2509.04439*, 2025.

[25] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, Mar. 2024.

[26] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

[27] Haochen Shi, Xingdi Yuan, and Bang Liu. Evolving programmatic skill networks. *arXiv preprint arXiv:2601.03509*, 2026.

[28] Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. Rlad: Training llms to discover abstractions for solving reasoning problems. *arXiv preprint arXiv:2510.02263*, 2025.

[29] Yiqing Xie, Emmy Liu, Gaokai Zhang, Nachiket Kotalwar, Shubham Gandhi, Sathwik Acharya, Xingyao Wang, Carolyn Rose, Graham Neubig, and Daniel Fried. Hybrid-gym: Training coding agents to generalize across tasks. *arXiv preprint arXiv:2602.16819*, 2026.

[30] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

[31] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[32] Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

[33] Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026.

[34] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

[35] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025.

[36] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7080–7106, 2026.

[37] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026.

[38] Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by llms, 2026.

[39] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023.

[40] Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025.

[41] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[42] Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[43] Mauro Vallati, Lukas Chrpa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, et al. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36(3):90–98, 2015.

[44] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 74325–74362. Curran Associates, Inc., 2024.

[45] Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark*, 2025.

[46] 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.

Table A.2: Summary of datasets used in evaluation.

Domain	Dataset	# Instances	Notes
Math Reasoning	HMMT Feb 2025	30	Competitive math problems
	HMMT Nov 2025	30	Competitive math problems
	HMMT Feb 2026	33	Competitive math problems
	HMMT (Total)	93	Competitive math problems
Code Generation	BigCodeBench Hard	148	Real-world programming problems
	LiveCodeBench v6 (Hard)	80	Competitive programming problems
Agentic Tasks	ScienceWorld	90	Scientific experiment tasks
	PDDL	60	Symbolic planning tasks

A Experimental Setup Details

A.1 Dataset Details

We evaluate our method on a diverse set of benchmarks spanning mathematical reasoning, code generation, and multi-turn agentic tasks. All datasets used in this work are publicly available.

For mathematical reasoning, we use exam problems from the `Harvard-MIT Mathematics Tournament (HMMT)`, an elite high-school competition featuring challenging mathematical reasoning questions spanning algebra, combinatorics, geometry, and number theory. Specifically, we combine problem sets from three recent competitions: February 2025, November 2025 and February 2026, all obtained from the `MathArena` collection on HuggingFace.[5] We evaluate performance using accuracy against ground-truth answers, following the official grading scripts provided by `MathArena`.

For code generation, we evaluate on the `BigCodeBench Hard` benchmark, a subset of BigCodeBench designed to assess real-world programming and compositional reasoning abilities,[6] and `LiveCodeBench`, a benchmark of competitive programming problems.[7] Performance is measured using pass@1-style accuracy (pass rate), i.e., whether the generated program passes all ground-truth (including hidden) test cases.

For evaluating multi-turn agentic behavior, we use tasks from the AgentBoard benchmark suite.[8] Specifically, we evaluate on `ScienceWorld`, a simulated environment requiring sequential decision-making and scientific reasoning, and `PDDL`, symbolic planning tasks defined in the Planning Domain Definition Language. Each task involves multi-step interaction with an environment, where the model generates actions conditioned on observations. Tasks are partially observable and require planning over multiple turns. We evaluate performance using the success rate (whether the task is completed) and progress rate (fraction of sub-goals achieved) provided by AgentBoard.

Table A.2 summarizes the information of each dataset.

A.2 Model Details

We use GPT-4o (version `2024-11-20`) [46] with a temperature of 0 and $top\_p = 0.5$ on BigCodeBench. On ScienceWorld and PDDL, we use o4-mini (version `2025-04-16`) with low reasoning effort.[9] On LiveCodeBench and HMMT, we use o4-mini (version `2025-04-16`) with high reasoning effort.

A.3 Baseline Details

We compare our method against representative test-time scaling (TTS) and test-time learning (TTL) methods. For TTS, we include Best-of-N sampling, which generates N independent candidate solutions per task and selects the one with the highest self-estimated score. We also evaluate Recursive Self-Aggregation (RSA) [6], a stronger TTS method that iteratively refines a population of reasoning trajectories by aggregating subsets of candidates, enabling progressive improvement across iterations. We run it with the population size $N = 16$ and aggregation subset size $K = 4$ with GPT-4o and o4-mini to balance between inference cost and performance. For TTL

[5] `https://huggingface.co/datasets/MathArena/hmmt_feb_2025`, `https://huggingface.co/datasets/MathArena/hmmt_nov_2025`, `https://huggingface.co/datasets/MathArena/hmmt_feb_2026`

[6] `https://huggingface.co/datasets/bigcode/bigcodebench-hard`

[7] `https://github.com/LiveCodeBench/LiveCodeBench` (we use the hard subset of v6).

[8] `https://hkust-nlp.github.io/agentboard/`

[9] `https://openai.com/index/o3-o4-mini-system-card/`

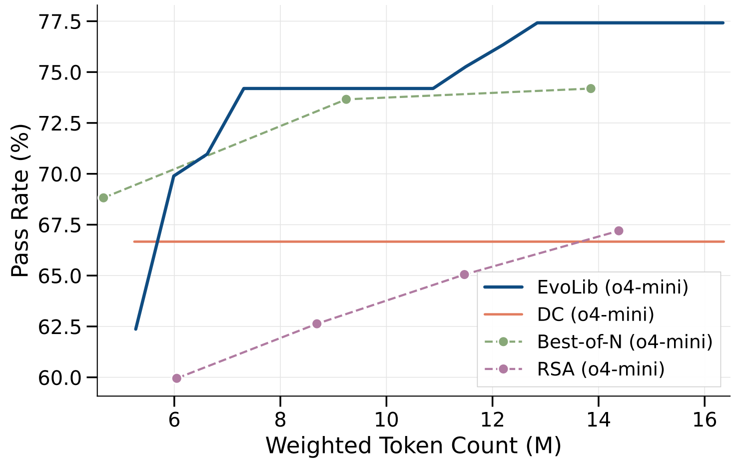


Figure B.1: Cost–performance curve comparing EvoLib with competitive baselines on HMMT. Each curve plots performance (y-axis) as the test-time compute cost (x-axis) increases for each method.

baselines, we include ExpRAG [9],[10] which augments the base model with a global memory of raw trajectories, and Dynamic Cheatsheet [36],[11] which maintains an adaptive memory (i.e. a cheatsheet) of high-level principles and procedural knowledge extracted through self-reflection. For all baselines, we run them with varying token cost budget (i.e. varying N for Best-of-N and varying number of iterations for the other methods) and compare performance under the same cost budget.

A.4 EvoLib Details

To estimate the baseline and conditional scores, the agent typically samples multiple times for each problem x_t, except for the continual learning experiments where the agent attempts each task only once. In this case, the agent uses samples at previous turns for the same task to estimate the scores. We set the number of trials $K = 3$ on code generation tasks, and $K = 10$ for mathematical reasoning, while on multi-turn agentic tasks, each task is attempted only once (following the continual learning setting). To retrieve similar functions for abstraction consolidation, we set the embedding-based similarity threshold to 0.8. For weighted sampling, we sample at most 10 insights and 10 skills into the LLM prompt, and set the task-abstraction similarity threshold to 0.2 on agentic tasks and to 0 on the other benchmarks (since the problems within each benchmark are highly related on these benchmarks). We use *text-embedding-3-small* as the embedding model.

B Extended Results

Cost-Performance Curve on HMMT. Fig. B.1 shows the cost–performance curve on HMMT. EvoLib maintains a consistent advantage across the compute range, similar to the trends observed on coding benchmarks in the main paper. Although EvoLib adds an average of 1-2 LLM calls per iteration for scoring, maintaining and updating the library, it improves performance more rapidly with fewer attempts than the TTS approaches.

Library growth and the role of consolidation. Fig. B.2 compares the growth of the library with and without consolidation. Without consolidation, the number of entries increases nearly linearly with the number of iterations, reflecting the accumulation of task-specific abstractions. In contrast, with consolidation, the growth is substantially slower at later stage. Case 3 in Table B.1 provides a concrete example of the consolidation process. A newly extracted function and an existing function are merged into a more general abstraction that subsumes both behaviors. This illustrates how the library evolves from specific, task-level functions into abstractions that apply across multiple problems. Such generalization is central to enabling transfer across tasks without explicit supervision. These results support the role of consolidation in controlling memory growth while encouraging the emergence of generic abstractions.

Evolution of abstraction quality. Fig. B.3 examines how the quality of the most useful abstractions evolves over time. Both the average IG and average Future IG among the top entries increase as the number of iterations grows. The increase in IG indicates that the library progressively contains abstractions that directly

[10] We set the number of retrieved entries to 4 as recommended and use *text-embedding-3-small* as the embedding model.

[11] We use the *DC-Cumulative* variant in all our experiments as it outperforms the other variant in the preliminary experiment.

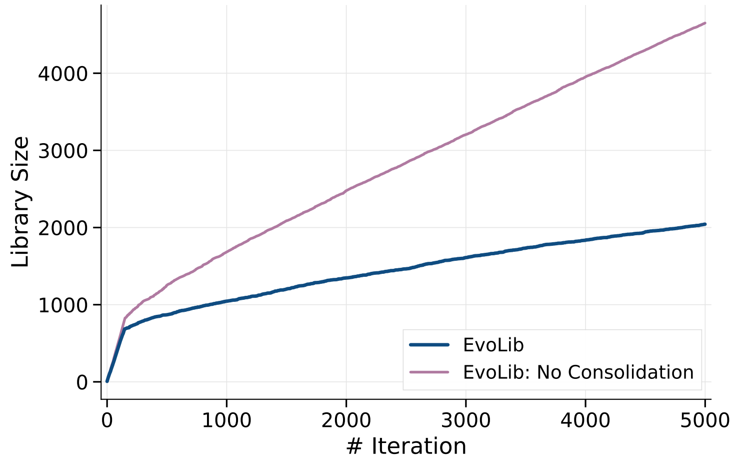
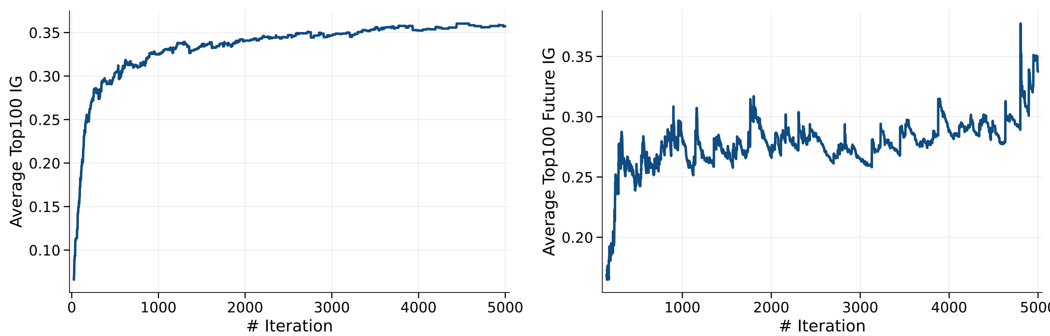


Figure B.2: Growth of the library size over iterations with and without consolidation on BigCodeBench. The x-axis shows the number of iterations and the y-axis shows the number of abstractions stored in the library.



(a) Average IG Among Top 100 Entries

(b) Average Future IG Among Top 100 Entries

Figure B.3: Average Information Gain (IG) and Future Information Gain (Future IG) among the top 100 abstractions in the library over iterations on BigCodeBench. The x-axis shows the number of iterations and the y-axis shows the average score for the top-ranked abstractions.

improve solution quality. The increase in Future IG suggests that the library also becomes better at generating abstractions that enable further learning. Together, these trends provide evidence that the weighting mechanism prioritizes abstractions that contribute both immediate and future utility.

Effect of task order in continual learning. Table B.2 compares EvoLib with Dynamic Cheatsheet (DC) under different task orders, including easy-to-hard, hard-to-easy, and random permutations. EvoLib consistently outperforms DC across all settings, achieving higher success rates and progress rates. Notably, the advantage is largest under random ordering, where no curriculum structure is present, suggesting that EvoLib is more robust to variations in the task stream.

Importantly, since PDDL comprises four different types of tasks that require different types of actions and planning strategies, strong performance under random ordering indicates that EvoLib can effectively handle and continually learn from a heterogeneous mixture of task types without relying on a structured curriculum. This property is desirable in real-world scenarios, where an agent must respond to diverse user requests in arbitrary order and still accumulate useful abstractions over time.

Overall, these results support the hypothesis that EvoLib enables continual learning that is less dependent on task ordering, whereas methods based on linear knowledge updates are more sensitive to the ordering.

EvoLib addresses the failure modes in continuously updated abstract memory. Recent work shows that continuously updated abstract memory in existing approaches can degrade performance through three major failure modes: *misgrouping*, where experiences from different types of tasks are consolidated into the

	Consolidation Case Study
1	**(Over-generalized → Grounded)** **Old Insight:** If you need to move multiple objects between two rooms and have two free grippers, then pick up two different objects in the source room before moving, so you can carry both in one trip. **New Insight:** If both grippers are free and there are objects in the same room that need relocation, then use both grippers simultaneously to pick objects, reducing the number of move–drop cycles. **Consolidated Insight:** If both grippers are free and there are multiple objects in the same room that need relocation, then pick up two different objects simultaneously before moving, so you can carry both in one trip and reduce the number of move–drop cycles.
2	**(Over-generalized → Grounded)** **Old Insight:** If a calculated metric such as AUC or any other value can result in NaN due to invalid inputs or edge cases, then handle such cases explicitly (e.g., skipping computation, default values, or raising errors). **New Insight:** If a metric (e.g., AUC) can result in NaN due to edge cases (e.g., all true labels are the same → undefined ROC), then explicitly detect and handle this condition. **Consolidated Insight:** If a calculated metric such as AUC or any other value can result in NaN due to invalid inputs or edge cases, such as all true labels being the same leading to an undefined ROC curve, then handle such cases explicitly (e.g., skipping computation, default values, or raising meaningful errors).

3	**(Overfitting → General)**
	Old Skill:
	`def validate_dataframe_columns(data, min_columns):`
	`  if not isinstance(data, pd.DataFrame):`
	`    raise ValueError(...)`
	`  if data.shape[1] < min_columns:`
	`    raise ValueError(...)`
	New Skill:
	`def validate_minimum_categories(data, columns, min_categories=2):`
	`  for col in columns:`
	`    if data[col].nunique() < min_categories:`
	`      raise ValueError(...)`
	Consolidated Skill:
	`def validate_dataframe(data, min_columns=None, column_category_constraints=None):`
	`  if not isinstance(data, pd.DataFrame):`
	`    raise ValueError(...)`
	`  if min_columns is not None:`
	`    if data.shape[1] < min_columns:`
	`      raise ValueError(...)`
	`  if column_category_constraints is not None:`
	`    for col, k in column_category_constraints.items():`
	`      if col not in data.columns:`
	`        raise ValueError(...)`
	`      if data[col].nunique() < k:`
	`        raise ValueError(...)`

Table B.1: Consolidation cases illustrating generalization and grounding effects in EvoLib.

same abstraction; *interference*, where repeated rewriting strips away applicability conditions and turns useful lessons into overly broad guidance; and *overfit*, where memory becomes tied to narrow surface patterns rather than reusable strategies [38]. By contrast, our experiments show that EVOLIB avoids or substantially alleviates these issues:

Misgrouping. EVOLIB is less susceptible to misgrouping because its consolidation is localized rather than global: two abstractions are merged only when they are close in the embedding space and are judged semantically and functionally similar. Table B.2 provides further evidence: EVOLIB remains strong even under random task ordering in PDDL, where four task types are interleaved, suggesting that the library can accumulate useful abstractions from a heterogeneous stream without requiring a clean curriculum to keep different task families separated.

Interference. The interference failure mode arises when repeated rewriting and consolidation remove the conditions under which a lesson is valid, causing an abstraction learned from one setting to mislead behavior in another. EVOLIB mitigates this effect as it extracts an abstraction from one trajectory at a time and only

Table B.2: Success and Progress Rate on PDDL under different task orders. We report mean ± standard deviation over three runs for each method.

Method	**Easy → Hard**		**Hard → Easy**		**Random**	
	Success	Progress	Success	Progress	Success	Progress
DC	66.7 ± 2.9	82.2 ± 2.5	65.6 ± 1.9	80.7 ± 0.6	65.6 ± 1.0	81.5 ± 0.8
EvoLib	**71.1 ± 2.5**	**85.5 ± 3.0**	**68.9 ± 4.8**	**86.3 ± 0.7**	**74.4 ± 1.9**	**88.1 ± 1.5**

consolidates it with another abstraction when they are semantically and functionally similar. Case 1 and 2 in Table B.1 show that the consolidation process in EvoLib can move abstractions in the opposite direction: it adds more grounded conditions to the insight statements rather than smoothing them away into generic advice. In addition, an abstraction that interferes with a broad range of tasks will receive low Future IG scores, decreasing its chance to be sampled during inference. As shown in Tables B.3 to B.7, the top high-Future-IG abstractions remain concrete and applicable to only a subset of tasks.

Overfit. The overfit failure mode occurs when memory tracks recurring surface regularities of past instances instead of extracting a strategy that transfers to related tasks. This problem is mitigated in EvoLib by the consolidation and weighting mechanism. An abstraction that is specific to a certain task can be consolidated with a new one (from a related but different task) into a more general abstraction, as shown by Case 3 in Table B.1. Furthermore, abstractions that remain overly specific will then receive low Future IG scores and thus low sampling weights. As shown in Tables B.3 to B.7, the highest-Future-IG entries are typically reusable procedures, checks, or reasoning templates that are applicable to similar tasks.

	Top Insights (BigCodeBench)
1	If (your function or program processes input that requires specific formatting or encoding, such as hexadecimal strings or UTF-8 encoded data), then (handle potential decoding errors gracefully by implementing checks and providing meaningful error messages before proceeding with further processing).
2	If a function depends on external systems (e.g., network, file I/O, or hardware) to produce results, then ensure to mock or simulate these external dependencies during testing to avoid false negatives caused by environmental factors or unhandled edge cases.
3	If (a function interacts with a directory or file system), then consider the case (the directory or file may not exist) and handle it explicitly by checking for existence before performing operations, or by using try-except blocks to catch and handle `FileNotFoundError` or similar exceptions.
4	If (a function processes external files or data inputs), then ensure to validate the existence, format, and content of the files or data before proceeding with further operations. Specifically, check for conditions such as file existence, non-empty content, and adherence to expected structure or format. Handle errors gracefully by providing meaningful error messages, consider edge cases like empty files or missing data to prevent runtime exceptions, raise appropriate exceptions or provide fallback mechanisms for invalid data, and explicitly handle cases where the input might be empty, corrupted, or invalid.
5	If your code involves calculating metrics like AUC or other statistical measures, then ensure that the input data (e.g., true labels and predicted probabilities) is valid and does not contain edge cases such as all true labels being the same class (e.g., all 0s or all 1s) or predicted probabilities being constant, as these can result in undefined or NaN values. Consider adding checks to validate the input data and handle such cases explicitly, such as skipping the calculation, providing a meaningful default value, or raising an informative error.
	Top Skills (BigCodeBench)
1	`def validate_range(range_low, range_high):` `""" Validate that the lower bound is less than the upper bound.  Parameters:  range_low (int):  Lower bound of the range.  range_high (int):  Upper bound of the range.  Raises: ValueError:  If range_low is not less than range_high.  """`
2	`def fetch_and_prepare_boston_data(data_url):` `""" Fetches and prepares the Boston Housing dataset from the given URL. Parameters: data_url (str):  URL to the Boston Housing dataset.  Returns:  pd.DataFrame:  A DataFrame containing the prepared Boston Housing data.  """`
3	`def task_func(kwargs, target_dir="non_none_files"):` `""" Process files from a dictionary by checking if the file exists, and if it has content, then copies it to a target directory.  Parameters:  kwargs (dict):  dictionary of file paths to content.  target_dir (str):  destination directory.  Returns:  list of copied file paths.  """`
4	`def task_func(df, target_column):` `""" Train a random forest classifier to perform classification and plot feature importances.  The plot uses feature importance scores on the x-axis and feature names on the y-axis, sorted in descending order.  Returns the trained model and plot Axes.  """`
5	`def encrypt_aes_key_with_rsa(aes_key, public_key):` `""" Encrypts an AES key using an RSA public key.  Parameters:  aes_key (bytes):  The AES key to encrypt.  public_key (rsa.PublicKey):  The RSA public key.  Returns:  bytes:  The encrypted AES key.  """`

Table B.3: Top insights and skills with high Future IG in EvoLib on BigCodeBench.

	Top Insights (LiveCodeBench)
1	If you're maintaining counts of available items by splitting them into "below" and "above" a moving threshold, then after picking an item you must decrement the count of whichever subgroup it came from (or adjust your pointer), not just the total used count, so your future availability calculations remain correct.
2	If you define a block in Python (such as a function, class, loop, or conditional) ending with a colon, then do include at least one indented statement (even just a `pass`) immediately after, otherwise you'll get an "expected an indented block" syntax error.
3	If you split your digit-DP into two passes (one for numbers that contain a special digit and one for numbers that don't), then do not treat every zero the same—use a "started" flag so that leading zeros aren't counted as actual zeros, only set your "have_zero" flag when you see a zero after you've "started," and in the zero-free pass forbid zeros once "started." This way your two DPs form a disjoint, exhaustive partition and you won't undercount or overcount any numbers.
4	If your problem involves multiple agents that cannot occupy the same vertex at the same time—but may visit the same vertex at different times—then do not model these conflicts as permanent vertex-capacity constraints. Instead, you must encode the temporal dimension (for example by building a time-expanded graph or by BFS on the joint (position,time) state).
5	If you're using binary search to maximize or minimize an integer parameter via a feasibility predicate, then verify that your initial bounds cover the entire range of possible values, ensure the predicate is monotonic, and test edge cases; otherwise you may cut off the real optimum or get the wrong threshold.
	Top Skills (LiveCodeBench)
1	`def greedy_interval_cover(sorted_intervals, cover_L, cover_R):` `""" Greedy interval covering algorithm. Given intervals sorted by left endpoint, selects a minimum-cardinality sub-collection that covers [cover_L, cover_R]. Returns selected indices or empty list if impossible. """`
2	`def divisors_from_prime_counts(prime_counts: dict) -> list:` `""" Generate all divisors of a number from its prime factorization using DFS over prime powers. """`
3	`def kmp_search(text, pattern):` `""" Return True if pattern occurs in text using the KMP algorithm. Time complexity: O(len(text) + len(pattern)). """`
4	`def tarjan_scc(graph):` `""" Compute strongly connected components of a directed graph using Tarjan's algorithm. Returns a list of components. """`
5	`def binary_search_last_true(lo, hi, pred):` `""" Given a monotonic predicate pred(x), find the maximum x in [lo, hi] such that pred(x) is True. """`

Table B.4: Top insights and skills with high Future IG in EvoLib on LiveCodeBench.

	Top Insights (HMMT)
1	If a single oriented half-space on two parallel convex polygons selects "sup–blocks" of vertices, then those blocks on the two polygons must share the same extremal index (they start at the same vertex) unless one of the blocks is empty or the entire polygon.
2	If you use inclusion–exclusion to count "bad" subsets defined by forbidden substructures (cuts, patterns, etc.), you must list and include all minimal forbidden substructures—in particular those of higher cardinality that do not contain smaller ones—to avoid undercounting.
3	If a point induces two triangles with a common "angle-ratio" condition and known side-length ratio, then view the situation as a spiral similarity centered at that point—write its rotation angle and scale factor explicitly, and impose the "sum of the four angles around a point equals 360°" to link the small triangle's base angle to the given interior angle.
4	If you use the Law of Sines in two triangles sharing no common angle, then a ratio of side-lengths such as BP/PC generally equals a product of three sine-ratios, not simply sin x/sin y. Always carry along the extra factors from both sine laws.
5	If you need the sign of sin(x) by comparing x to multiples of π, then use floor(x/π) mod 2: sin(x)$>$0 exactly when floor(x/π) is even. Moreover, parity (floor($2^n \cdot \alpha$)) equals the nth binary digit of α.
	Top Skills (HMMT)
1	**Eliminate variables by combining constraints:** When you have two constraints (e.g., a right-angle/power relation plus a circle/quadratic relation), substitute one into the other to eliminate a squared term and derive a simpler (often linear) relationship among the remaining variables.
2	**Exploit symmetry to reuse a count:** If a configuration has an involutive symmetry (swapping roles, reversing a half-space, etc.), map the "opposite case" to the original one to conclude both cases contribute equally, rather than recounting from scratch.
3	**Re-encode divisibility as a product order:** For sets defined by prime-power structure, represent elements by exponent vectors; then divisibility becomes coordinatewise comparison, turning the problem into reasoning about a product of chains (a grid poset).
4	**Parameterize points on a segment affinely:** To locate points at specified distances along a segment, express each point as an affine combination $P = (1-t)B + tC$ with t equal to the normalized arc-length fraction; then read off coordinates or derived quantities cleanly.
5	**Reduce interacting motion to a standard stochastic process:** When multiple moving objects merge upon contact, reinterpret the dynamics as coalescing random walks (or an equivalent Markov process) on an appropriate state space, enabling use of known results or simpler expectation calculations.

Table B.5: Top insights and skills with high Future IG in EVOLIB on HMMT.

	Top Insights (ScienceWorld)
1	If you are navigating to a known target location, then first confirm the correct connecting doorway by using "look around" in your current room, plan the most direct route, and avoid back-and-forth movements that don't add information.
2	If you need to choose among multiple objects, first list or note all candidates, then compare their key attributes (e.g. life span) before focusing on one.
3	If you intend to select from an ambiguity menu, do not type just the index; you must include the action keyword plus the exact indexed object name.
4	If you have heated an ingredient and see "Progress 0.5/1.0," then check with the appropriate measuring device to complete the task.
5	If a substance must reach a phase change, then ensure the heating source is fully activated and give sufficient waiting time (or confirm rising temperature) before checking for melting.
	Top Skills (ScienceWorld)

1	**Functionality:** Heat a substance in a container using the stove until it reaches or exceeds a target temperature.
	Condition: When you need to raise the temperature of whatever is in the vessel to a desired threshold.
	Input Variables: Stove, Thermometer, Vessel, TargetTemp.
	Steps:
	`move Vessel to Stove`
	`activate Stove`
	`wait1`
	`wait1`
	`wait1`
	`use Thermometer on Vessel`
	`(if temperature < TargetTemp, repeat wait and measure)`
	`deactivate Stove`

2	**Functionality:** Deliver or deposit an object from your inventory into a target container.
	Condition: When you have the object in inventory and need to place it into a named container.
	Input Variables: OBJ, DEST_LOC, CONTAINER.
	Steps:
	`(if not at DEST_LOC) go to DEST_LOC`
	`look around`
	`move OBJ to CONTAINER`

3	**Functionality:** Retrieve an item from a storage container.
	Condition: When an object needed for a task is inside a container.
	Input Variables: OBJ, CONTAINER.
	Steps:
	`open CONTAINER`
	`look in CONTAINER`
	`pick up OBJ`

4	**Functionality:** Verify the physical state of a substance after a process.
	Condition: After performing a process intended to change a substance's state (e.g., heating or freezing).
	Input Variables: STORAGE, CONTAINER, SUBSTANCE.
	Steps:
	`(if STORAGE exists) open STORAGE`
	`open CONTAINER`
	`look in CONTAINER`
	`focus on SUBSTANCE`
	`look at SUBSTANCE`

5	**Functionality:** Wait until an object changes state over time.
	Condition: When an environmental or physical process requires time to complete.
	Input Variables: desired_state.
	Steps:
	`wait`
	`(or) wait1`
	`look around`
	`repeat until state changes`

Table B.6: Top insights and skills with high Future IG in EvoLib on ScienceWorld.

	Top Insights (PDDL)
1	If working with nuts on a hub, then jack up the hub before undoing or removing the wheel; after wheel removal or installation, refasten the nuts and jack down; and ensure the hub is on the ground before tightening any loose nut.
2	If you leave a shot glass on the table after cleaning, then you can pour into it directly from a shaker without grasping it, as long as you satisfy the hand-holding requirements of the pour action.
3	If you finish dropping the last carried object, then check immediately for any remaining objects in the other room to avoid idle moves.
4	If the hub is on the ground, then you must loosen the nuts before jacking up; otherwise removal is impossible.
5	If a block you want is not on the table, then unstack it from whichever block it's on before attempting to pick it up.
	Top Skills (PDDL)

1	**Functionality:** Load two specified ingredients into a clean shaker using two separate shot glasses.
	Condition: The shaker is clean and empty; both shot glasses are clean and empty; both hands are empty; dispensers are available.
	Input Variables: shot1, ingredient1, shot2, ingredient2, hand, other_hand, shaker, dispenser1, dispenser2, l0, l1, l2.
	Steps:
	`hand grasp shot1`
	`fill-shot shot1 ingredient1 hand other_hand dispenser1`
	`pour-shot-to-clean-shaker shot1 ingredient1 shaker hand l0 l1`
	`clean-shot shot1 ingredient1 hand other_hand`
	`fill-shot shot2 ingredient2 hand other_hand dispenser2`
	`pour-shot-to-used-shaker shot2 ingredient2 shaker hand l1 l2`
	`hand leave shot2`

2	**Functionality:** Remove a wheel from a hub and stow it in the boot.
	Condition: The hub is on the ground, the nut is tight, tools are available, and the boot is open.
	Input Variables: hub, wheel, nut, boot.
	Steps:
	`loosen nut hub`
	`jack-up hub`
	`undo nut hub`
	`remove-wheel wheel hub`
	`put-away wheel boot`
	`jack-down hub`

3	**Functionality:** Fetch the basic tools (wrench and jack) from a closed container.
	Condition: The container is closed but unlocked, and tools are inside.
	Input Variables: container, wrench, jack.
	Steps:
	`open container`
	`fetch wrench container`
	`fetch jack container`

4	**Functionality:** Move a block from one block to another.
	Condition: The arm is empty; block X is clear and on block U; block V is clear.
	Input Variables: X, U, V.
	Steps:
	`unstack X U`
	`stack X V`

5	**Functionality:** Shake a two-ingredient mixture in a shaker to form a cocktail.
	Condition: The shaker contains the correct ingredients and is unshaken.
	Input Variables: cocktail, ingredient1, ingredient2, shaker, hand1, hand2.
	Steps:
	`hand1 grasp shaker`
	`shake cocktail ingredient1 ingredient2 shaker hand1 hand2`

Table B.7: Top insights and skills with high Future IG in EvoLib on PDDL.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contribution (EvoLib) and its scope, including knowledge accumulation without parameter updates and empirical improvements across benchmarks (Sections 1 and Abstract).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated discussion of limitations (Section 5, "Limitations").

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include any theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.

- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the algorithm, experimental setup, datasets, baselines, and evaluation metrics in sufficient detail to reproduce results, including pseudocode (Algorithm 1), method and dataset descriptions (Section 4 and Appendix A). The full code (including prompts) is released in the supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The full code (including prompts) is released in the supplementary material. All datasets used are referenced.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all the dataset, model, hyperparameter, evaluation, and configuration details in Sections 4 and Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report statistical significance of the main result in Table 1 and explain how they were calculated.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Figure 2 and B.1 shows the amount of compute needed (in token count) to reach varying levels of performance for each method on each dataset.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research adheres to standard machine learning experimentation practices, uses public datasets, and does not involve human subjects, sensitive data or harmful applications as presented.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 5 explicitly discusses this work as a stepping stone toward self-improving agents, which is a positive societal impact on effective AI assistant. The "Limitations" paragraph further discusses potential negative impact when there is a misalignment between human and LLMs' judgment.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work does not release new high-risk models or datasets and does not involve potentially sensitive or misuse-prone assets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets, baseline methods, and models used are cited and used in accordance with their license.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The full code together with clear documentation are released in the supplementary material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The research does not involve human subjects or crowdsourced data collection.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The study does not involve human participants and therefore does not require IRB approval.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of the method (used for solution generation, evaluation, abstraction extraction, and consolidation), and their role is explicitly described throughout the paper (Sections 3 and 4).

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.