OmniReflect: Discovering Transferable Constitutions for LLM agents via Neuro-Symbolic Reflections

Anonymous ACL submission

Abstract

Efforts to improve LLM agent performance on complex tasks have largely focused on fine-tuning and iterative self-correction. However, these approaches often lack generalizable mechanisms for long-term learning and remain inefficient in dynamic environments. We introduce OmniReflect, a hierarchical reflection-driven framework that constructs constitutions, compact sets of guiding principles distilled from past task experiences, to enhance the effectiveness and efficiency of LLM agents. OmniReflect operates in two modes: Self-sustaining, where a single agent periodically curates its own reflections during task execution, and Co-operative, where a meta-advisor derives constitutions from a small calibration set to guide another agent. To construct these constitutional principles, we employ Neural, Symbolic, and Neuro-Symbolic techniques, offering a balance between contextual adaptability and computational efficiency. Empirical results averaged across models show major improvements in task success, with absolute gains of +10.3% on ALFWorld, +23.8% on BabyAI, and +8.3% on PDDL in the Self-sustaining mode. Similar gains are seen in the Co-operative mode, where a lightweight Qwen3-4B ReAct agent outperforms all Reflexion baselines on BabyAI. These findings highlight the robustness and effectiveness of OmniReflect across environments and backbones.

1 Introduction

Foundational Large Language Models (LLMs) are increasingly being deployed as autonomous agents to explore environments and utilize tools for executing complex, often compound tasks on behalf of users (Shen et al., 2024; Yang et al., 2023; Nakajima, 2023). Despite these advancements, recent studies continue to highlight key limitations of LLM-based agents, particularly in reasoning, planning, and continual learning (Jain et al., 2024; Huang et al., 2023).

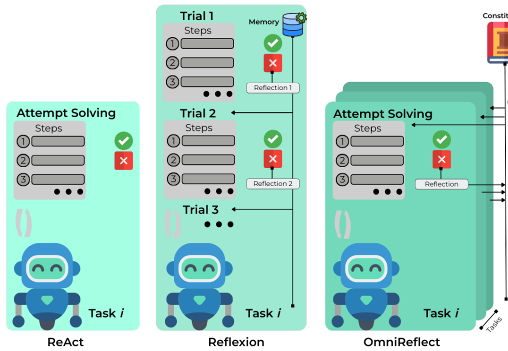


Figure 1: Existing strategies such as ReAct (Yao et al., 2022) rely on step-by-step reasoning within a single trajectory, while Reflexion (Shinn et al., 2023) enhances performance through iterative retries guided by self-critiques. In contrast, OmniReflect maintains categorized constitution rules (summarized reflections) that guide task-solving by collating knowledge over time.

Two major strategies are commonly employed to mitigate these limitations:

1. **Fine-tuning**, typically performed on large-scale, environment-specific datasets using supervised or reinforcement learning approaches (Guo et al., 2025; Deng et al., 2024; Shridhar et al., 2020), which is computationally expensive and often lacks scalability.

2. **Reasoning and self-correction**, where an LLM agent is prompted step-by-step to reason about its actions and revise its course if the task remains incomplete (Shinn et al., 2023; Madaan et al., 2023; Xi et al., 2023).

Self-correcting methods usually involve analyzing and critiquing failed attempts to extract task-specific feedback (called reflection) that can guide future trials (Shinn et al., 2024; Madaan et al., 2023). Although these approaches enhance task-level performance, they often fall short in achieving abstraction and generalization in an environment (Xie et al., 2024).

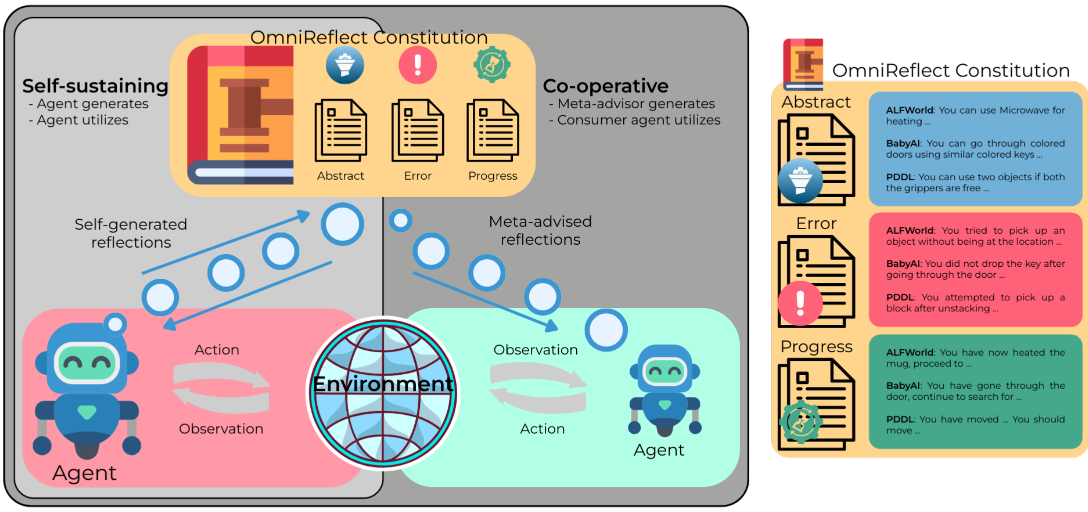


Figure 2: The OmniReflect framework operates in two reflective modes: (1) Self-Sustaining: where the LLM agent periodically generates, updates and uses constitutions; and (2) Co-operative: where a meta-advisor derives constitution rules from a calibration set to guide another agent. These rules encode task-level and environment level, knowledge for effective task completion. Examples are shown on the right.

We introduce OmniReflect, a hierarchical reflection-based framework that adaptively curates an evolving *constitution*, a set of categorized and summarized reflections, to guide task-solving by accumulating knowledge over time. The constitution serves as structured memory, distilled from prior task experiences within an environment. By tightly integrating reflections with task execution, OmniReflect not only enables efficient task completion but also facilitates the creation of reusable knowledge. To ensure the continued utility and coherence of constitution (e.g., by reducing redundancy), OmniReflect performs periodic summarization, transforming unstructured episodic traces into well-organized rule book. Figure 1 provides an overview of the OmniReflect framework and highlights key differences compared to existing approaches.

OmniReflect can be used in two distinct modes: 1) Self-sustaining and 2) Co-operative. Figure 2 illustrates how both modes generate and utilize constitution, structured reflections accumulated across the environment. In the Self-sustaining mode, the same LLM agent periodically generates and reuses summarized reflections, curating them at fixed step intervals. In the Co-operative mode, a meta-advisor agent uses a small calibration set of tasks to generate constitution rules through reflections, which are then consumed by a separate task-solving agent. This modular integration leads to consistent performance improvements with minimal computational overhead. Furthermore, the synergistic integration of these two modes creates a robust system that acquires foundational knowledge during post-training calibration and incrementally refines it at test time, complementing both fine-tuning and self-correction strategies.

We leverage Neural, Symbolic, and Neuro-Symbolic techniques to construct constitutions. Neural methods offer rich, context-aware feedback but are often computationally intensive. Symbolic methods, while efficient and interpretable, tend to lack flexibility. Our Neuro-Symbolic approach strikes a balance, combining the adaptability of neural models with the structural efficiency of symbolic reasoning to enable scalable and cost-effective constitution generation. We empirically validate this design across diverse agentic benchmarks and LLMs, demonstrating that OmniReflect consistently outperforms strong baselines while solving tasks in fewer steps, highlighting both its effectiveness and efficiency.
Our key contributions are as follows:

- We propose OmniReflect, a hierarchical, reflection-driven agent framework that enables LLMs to accumulate reusable knowledge by adaptively creating and summarizing constitutions in parallel with task execution, thereby enhancing reasoning efficiency and overall agent performance.
- We demonstrate the robustness of OmniReflect's constitution-building in two modes (Self-sustaining and Co-operative)

showing strong performance even with minimal calibration data.

- We evaluate OmniReflect across diverse agentic benchmarks (ALFWorld, BabyAI, and PDDL) using multiple LLM backbones, consistently outperforming competitive baselines in task success and efficiency.

2 OmniReflect

The core design of our adaptable reflection framework, OmniReflect, is shown in Figure 2, with step-by-step operations described in Algorithm 1.

OmniReflect is designed to solve tasks from a dataset D by leveraging an LLM agent $\mathcal{L}$ that alternates between two complementary phases: (1) ACTION ($\mathcal{L}_{\text{a}}$) and (2) REFLECTION ($\mathcal{L}_{\text{ref}}$). In the ACTION phase, the agent generates an action conditioned on the task description, the current trajectory of steps taken, and the constitutions ($OmniC$) curated up to the previous REFLECTION phase. In the REFLECTION phase, the agent produces reflections based on the task description, the observed trajectory, and the existing constitution. These reflections include both task-agnostic long-term memories, shared across the environment, and task-specific progress updates, which act as short-term guidance for the current task. Section 2.2 provides a detailed breakdown of these reflection types.

Actions are generated at each turn until either a terminal state is reached or a predefined maximum number of steps ($\text{turns}_{\text{max}}$) is exceeded. In contrast, reflections are generated at fixed intervals determined by a reflection frequency hyperparameter (r_{freq}), while long-term memory summarization is triggered based on a separate summarization frequency (s_{freq}). We adopt a hyperparameter-driven approach, rather than automating the invocation via LLM-as-a-Judge strategies (Zheng et al., 2023; Gu et al., 2024), due to two main reasons: (1) It avoids the inefficiencies of known pitfalls in LLM-as-a-judge based automation (Jain et al., 2024; Gu et al., 2024; Li et al., 2025; Szymanski et al., 2025);(2) It offers greater control over the cost–performance trade-off of reflection.

Specifically, agents might invoke reflection either excessively (seeking frequent reassurance) or insufficiently, due to misplaced confidence. Additionally, randomization of test episodes across benchmarks can skew task-specific insight distributions and complicate automated summarization timing, potentially causing under- or over-summarization. We provide a detailed analysis of the effects of these hyper-parameters in Section 3.5.

Algorithm 1 OmniReflect Methodology

Input:
- ▶ Dataset D with set of tasks t_i, $D = \{t_i \mid i \in \{1, \dots, N\}\}$
- ▶ LLM Agent: $\mathcal{L}_a$, Reflection Agent: $\mathcal{L}_{\text{ref}}$
- ▶ Environment E producing ($observation$, $reward$) on receiving $action$

Output:
- ▶ Success Rate, SR
- ▶ Constitutions, $OmniC$

Initialize constitutions, $OmniC$
Intialize rewards, $rewards$
Initialize hyper-parameters, $r_{\text{freq}}, s_{\text{freq}}, \text{turns}_{\text{max}}$
for each task t_i in D **do**
  Initialize trajectory, τ
  Initialize task description, d_{t_i}
  Set turn $\leftarrow 0$
  while (turn $<$ $\text{turns}_{\text{max}}$) **do**
    ▷ *Action Phase*
    $act \leftarrow \mathcal{L}_a(d_{t_i}, OmniC, \tau)$
    $obs, reward \leftarrow E(act)$
    Append (act, obs) to τ
    ▷ *Periodic Reflection Phase*
    if (turn $\%$ r_{freq} is 0) **then**
      $rules \leftarrow \mathcal{L}_{\text{ref}}(d_{t_i}, OmniC, \tau)$
      Append $rules$ to $OmniC$
    end if
    if obs is final state **then**
      ▷ *Task is complete*
      break
    end if
    Increment turn
  end while
  Append $reward$ to $rewards$
  ▷ *Periodic Summarization*
  if (i $\%$ s_{freq} is 0) **then**
    $OmniC \leftarrow \mathcal{L}_{\text{ref}}(OmniC)$
  end if
end for
▷ *Compute success rate*
$SR \leftarrow mean(rewards) * 100$
return $SR, OmniC$

2.1 Reflection generation strategies

In this work, we explore three distinct approaches for generating natural language reflections based constitutions: 1) Neural 2) Symbolic and 3) Neuro-

Symbolic.

2.1.1 Neural Generation

Following prior work (Shinn et al., 2024; Madaan et al., 2024), we use LLMs to generate reflections. We craft reflection-oriented prompts and use $\mathcal{L}_{\text{ref}}$ to produce a list of rules or insights, which collectively form the constitution. To avoid redundancy and ensure meaningful information gain, $\mathcal{L}_{\text{ref}}$ also periodically summarizes past reflections. The constitution is continuously updated during task execution and leveraged in subsequent steps to guide the agent towards successful task completion.

2.1.2 Symbolic Generation

We extend the base regular expressions introduced by AgentBoard (Ma et al., 2024) for progress tracking, enhancing them to support both fine-grained and high-level template-based natural language feedback. Additionally, we design a concise set of task-specific and environment-level rules based on analysis of representative successful and failed trajectories. During each reflection step, the current trajectory, particularly the most recent observation, is evaluated against these regular expressions to generate reflections.

2.1.3 Neuro-Symbolic Generation

This method guides the Neural system using few-shot exemplars, which are derived from reflections produced during the Symbolic Generation process. This approach aligns LLM-generated reflections more closely with human intuition, while substantially reducing the need for extensive annotations typically required by reliable symbolic systems.

Refer to Appendix Section E for prompt details and Section F for sample regular expressions used across benchmarks.

2.2 Reflection categories

The REFLECTION phase in OmniReflect produces three distinct types of reflections: (1) **Abstract**, (2) **Error**, and (3) **Progress**. Figure 2 presents representative examples of each of these reflection types, drawn from sample tasks in the ALFWorld, BabyAI, and PDDL environments.

In the OmniReflect framework, abstract and error reflections are periodically summarized and stored in long-term memory, which is shared across tasks and sessions[1] at the environment level. In contrast, progress tracking reflections serve as transient, short-term knowledge that guides decision-making within the current task episode[2].

2.2.1 Abstract Reflections

We generate both *task-specific* and *task-agnostic* abstract reflections. In the *task-specific* setting, the agent solves individual tasks by using reflection to align its existing knowledge with the unique aspects of the current environment. This helps identify and correct errors caused by mismatched assumptions about the environment, often caused due to the implicit world knowledge of the model. In the *task-agnostic* setting, the agent's objective is to explore the environment broadly to generate reusable environment-level knowledge that benefits future tasks. In the OmniReflect framework, *task-specific* abstract reflections are generated in both the Self-sustaining and Co-operative modes. In contrast, *task-agnostic* abstraction is utilized exclusively in the Co-operative mode to address the challenge of limited data availability.

2.2.2 Error Reflection

We build upon prior work such as Reflexion (Shinn et al., 2024), extending its strategy of trajectory analysis to generate actionable feedback for error correction. Unlike existing methods that typically generate reflections post hoc or across multiple task trials (Renze and Guven, 2024), our OmniReflect framework introduces *in-situ* reflections: guidance is generated at periodic intervals within a single task execution episode. This enables the agent to recover from mistakes more rapidly, reducing the number of iterations needed for successful task completion. Furthermore, the agent evaluates the efficiency of its current trajectories, identifying recurring planning inefficiencies along with potential strategies for improvement.

2.2.3 Progress Reflection

OmniReflect generates task-specific progress-tracking thoughts, to assist in monitoring the execution of necessary sub-tasks in an optimal order to complete tasks. Unlike error reflections, they offer not only actionable guidance but also non-actionable, grounding observations that affirm completed sub-tasks, thereby reinforcing the agent's confidence in progressing to subsequent steps.

[1] While OmniReflect is initially designed for single-trial operation per task, it can be extented to multiple trials while preserving consistent knowledge sharing across trials.

[2] For clarity and conciseness, three types of reflections are not mentioned within the algorithm.

3 Experimental Details and Results

We conducted experiments on ALFWorld (Shridhar et al., 2020), BabyAI (Chevalier-Boisvert et al., 2018), and PDDL (Silver and Chitnis, 2020), three benchmark environments that balance navigation, reasoning, and compound task-solving. Details of the publicly available datasets, model configurations, and experimental settings are provided in the following sections.

3.1 Datasets

ALFWorld is designed to evaluate the ability of an agent to perform household tasks through goal-directed navigation and interaction. We use the *unseen split* of the dataset for our experiments. A detailed description is provided in Appendix B.

BabyAI is a grid-based environment in which agents navigate through interconnected minigrids to solve tasks such as "pick up a red box and then go through the grey door to the right". We adopt the same test split used by AgentBoard (Ma et al., 2024). See Appendix C for more details.

PDDL contains planning benchmarks focused on task decomposition and state optimization. Our experiments include four domains: *Gripper* (object transport between rooms), *Blocksworld* (block stacking and unstacking), *Barman* (cocktail preparation), and *Tyreworld* (tool-based mechanical repairs). We follow the same test split as used by AgentBoard. Refer to Appendix D for details.

3.2 LLMs and Experimental Setup

Large Language Models. Our study employed three widely recognized LLMs as agents: (1) Qwen3-4B (Team, 2025), (2) Gemini-2.0 (including its "*flash*" variant) and (3) GPT-4 (including its "*omni*" variant) (Hurst et al., 2024). To ensure a balance between reproducibility and performance, we set the temperature to 0, the nucleus sampling probability (*top_p*) to 0.7, the token sampling limit (*top_k*) to 50, and applied a repetition penalty of 1.

ReAct. The ReAct reasoning strategy (Yao et al., 2022) combines reasoning (thinking) and acting within agentic environments, enabling step-by-step decomposition of complex tasks. For ALFWorld, we use the prompts provided by the original authors (Yao et al., 2022), while for BabyAI and PDDL, we adopt ReAct prompts crafted by AgentBoard. Each experiment uses a single trial with a maximum of 50 turns per task.

Reflexion. While using Reflexion (Shinn et al., 2024) as our baseline, we adopt the same protocol as the original work, allowing up to 15 trials per task. Each trial consists of 50 turns.

OmniReflect. We adapt the ReAct reasoning strategy with 1-2 few-shot examples to construct our base prompt. For the primary results reported in Table 1, we use a reflection frequency (r_{freq}) of 10 and a summarization interval (s_{freq}) of 10 as hyper-parameters. Specifically, we generate reflections and update the constitution every 10 turns, and perform a constitution summary after completing 10 tasks. Similar to ReAct, OmniReflect uses a single trial with a maximum of 50 turns per task.

3.3 Self-sustaining mode: OmniReflect as an agent

The primary results for the self-sustaining mode are shown in Table 1. OmniReflect achieves the highest success rate in **7 out of 9** evaluated settings, outperforming Reflexion baselines under at least one of the neural, symbolic, or neuro-symbolic configurations. The only exceptions are Reflexion Gemini-2.0 on ALFWorld and Reflexion GPT-4 on PDDL. Averaged across models, OmniReflect yields substantial performance gains over Reflexion's 15-trial setup, despite operating with only a single trial augmented by periodic reflection: +10.3% on ALFWorld, +23.8% on BabyAI, and +8.3% on PDDL.

Performance patterns across environments further illuminate the strengths of each reflection strategy. On ALFWorld and PDDL, where tasks exhibit procedural regularities and structured action sequences, Symbolic and Neuro-Symbolic variants of OmniReflect consistently achieve top performance. These results highlight the strength of symbolic mechanisms, such as regular expressions, for progress monitoring and ensuring plan adherence. Conversely, in BabyAI, where success is tightly coupled to dynamic spatial exploration (e.g., object and door placement is random), Neural approaches dominate. This suggests that flexible, open-ended reasoning is better suited for environments with high variability and partial observability.

Collectively, these results underscore the efficacy of hierarchical, in-session reflection during task execution, demonstrating its ability to enable early identification and resolution of errors, leading

	ALFWorld			BabyAI			PDDL		
	Qwen	Gemini	GPT-4	Qwen	Gemini	GPT-4	Qwen	Gemini	GPT-4
ReAct	44.0	68.0	54.2	24.6	45.2	31.2	1.7	41.7	76.7
Reflexion	82.8	**94.0**	84.1	44.1	53.2	50.8	11.7	66.7	**91.7**
Self-sustaining mode									
OmniReflect-Neural	83.6	91.8	94.8	**73.2**	**74.1**	**72.3**	20.0	71.67	78.3
OmniReflect-Symbolic	**91.8**	88.8	**100.0**	45.5	67.9	60.7	16.7	**78.3**	**85.0**
OmniReflect-Neuro-Symbolic	86.6	**93.3**	96.3	54.5	64.3	68.8	**31.7**	75.0	80.0
Co-operative mode with ReAct agent									
OmniReflect meta-advisor$_{\text{Qwen}}$	73.1	77.6	93.8	58.0	58.0	59.8	12.1	36.6	70.0
OmniReflect meta-advisor$_{\text{Gemini}}$	55.2	47.0	94.8	50.9	58.0	52.7	10.0	45.7	75.0
OmniReflect meta-advisor$_{\text{GPT-4}}$	**76.9**	**79.1**	**96.3**	**60.7**	**63.4**	**64.2**	**13.3**	**65.2**	**80.0**

Table 1: Success Rate (%) of different LLM-agents across ALFWorld, BabyAI, and PDDL environments. All results follow the experimental setup described in Section 3.2. Reflexion (Shinn et al., 2023) results indicate final performance after 15 trials are completed. All ReAct and OmniReflect results only use 1 trial. The highest-performing results are shown in bold. Qwen and Gemini refer to Qwen3-4B and Gemini-2.0 respectively.

to significantly improved task completion across diverse environments.

3.4 Co-operative mode: OmniReflect as a Meta-Advisor

Table 1 also reports success rate improvements achieved when using OmniReflect meta-advisor models. In this setup, the meta-advisor constructs the constitution, while the consumer agent applies ReAct-style reasoning guided by the derived rules. When equipped with constitutions distilled via OmniReflect from just one calibration example per task type, ReAct agents exhibit substantial performance gains, achieving average improvements of 28% on ALFWorld, 29% on BabyAI, and 20.9% on PDDL, using GPT-4 as the meta-advisor. Crucially, these gains are realized without any additional LLM inferences at test time. Instead, the meta-advisor-generated constitutions are injected into the agent's prompt, demonstrating that even a lightweight integration of natural language guidance can yield strong downstream benefits. Notably, this setting utilizes only environment-level reflections (specifically abstract and error-level constitutions), without incorporating task-level progress tracking or dynamic reflection, as the ReAct agent does not performing on-the-fly reflective updates during task-execution.

The results underscore the effectiveness and transferability of OmniReflect-generated constitutions ($OmniC$), across different LLM backbones highlighting their scalability and versatility.

3.5 Ablation Studies and Discussion

Choice of OmniReflect Meta-Advisor. As illustrated in Table 1, reasoning capabilities are strongly correlated with both its ability to generate high-quality constitutions and to follow them effectively. A larger and more capable model such as GPT-4 demonstrates exceptional performance as both a meta-advisor (Section 3.4) and as a follower. Despite its smaller scale, Qwen3-4B proves to be a surprisingly competitive meta-advisor, frequently enabling greater performance gains in downstream ReAct agents compared to Gemini-2.0.

In total, **24 out of 27** evaluated meta-advisor/ReAct agent configurations show substantial performance improvements over the ReAct-only baseline, demonstrating the robustness and transferability of OmniReflect-derived constitutions across model architectures. The only exceptions occur when Gemini-2.0 serves as both the ReAct agent and meta-advisor on ALFWorld, and when GPT-4 ReAct agent is paired with either Qwen3-4B or Gemini-2.0 as meta-advisors on PDDL. These cases likely reflect either insufficient abstraction quality or weaker synergy in reflection transfer across model scales.

Notably, one compelling result is that a Qwen3-4B ReAct agent, when guided by GPT-4 as a meta-advisor, outperforms all Reflexion baselines on BabyAI. This highlights the potential for smaller models to exhibit advanced reasoning behavior when grounded with high-quality constitutions produced a capable OmniReflect Agent.

Impact of Reflection Hyper-Parameters. Table 2 illustrates that across all datasets, over-summarization can degrade performance, potentially omitting useful information. In contrast, increasing the frequency of reflection generally yields greater benefits, particularly in partially observable environments like BabyAI, where ongo-

$(r_{\text{freq}}, s_{\text{freq}})$	**ALFWorld**	**BabyAI**	**PDDL**
(5, 5)	95.2	67.8	73.3
(5, 10)	96.7	65.2	76.7
(5, 20)	**97.0**	65.2	78.3
(10, 5)	93.2	62.5	76.6
(10, 10)	94.8	**72.3**	**78.3**
(10, 20)	94.8	71.8	78.3

Table 2: Success rates on ALFWorld, BabyAI, and PDDL using the OmniReflect-Neural setting with GPT-4, illustrating the impact of reflection and summarization hyperparameters on task performance.

	ALFWorld	**BabyAI**	**PDDL**
ReAct	96.3	64.2	80.0
Neural	**100**	**78.5**	86.7
Neuro-Symbolic	**100**	74.1	**90.0**

Table 3: Success rate (%) on ALFWorld, BabyAI, and PDDL using OmniReflect-Neural and OmniReflect-Neuro-Symbolic settings with GPT-4, highlighting the added benefit of leveraging pre-generated constitutions from a GPT-4-based meta-advisor.

ing self-analysis helps the agent better assess its progress and adapt rapidly. This experiment uses only the OmniReflect-Neural setting, due to its minimal dependence on human annotations, making it more adaptable in practice. Additionally, the Symbolic variant typically do not perform periodic reflection; instead, they trigger reflection conditionally, based on the agent's current progress.

OmniReflect Agent with OmniReflect Meta-Advisor. Table 3 demonstrates that both neural and neuro-symbolic variants of the OmniReflect agent consistently outperform their ReAct counterparts, despite receiving identical guidance from the GPT-4 meta-advisor to mitigate the cold-start challenge. This indicates that OmniReflect agents not only effectively integrate external advice but also adapt and refine it over time, exhibiting a robust capacity to evolve initial guidance into more performant strategies.

Cost Efficiency. Figure 3 highlights the efficiency of the OmniReflect approach compared to both the baseline Reflexion method and standard ReAct agents. Across all datasets, OmniReflect consistently achieves performance comparable to or exceeding Reflexion, while maintaining inference efficiency on par with ReAct. Specifically, the bubble sizes (representing the average number of interaction turns) show that OmniReflect agents, regardless of backbone size, operate with efficiency comparable to ReAct, while achieving substantially higher success rates across a range of scenarios. Furthermore, OmniReflect's Neural and Neuro-Symbolic setting, when using multiple trials on ALFWorld (where Reflexion is the strongest) reach to 100% in just one additional trial. These results underscore the inefficiencies introduced by task-specific, trial-level reflections and siloed knowledge[3]. In contrast, OmniReflect's hierarchical, environment-level reflection framework enables more generalizable and cost-effective learning, positioning it as a scalable and effective alternative for LLM-based agents.

Reflexion may requires up to 764 LLM calls per task, while our neural approach reduces this to under 80, achieving $\sim$ 700 fewer inferences. OmniReflect-Symbolic does not require any additional LLM calls to generate reflections.

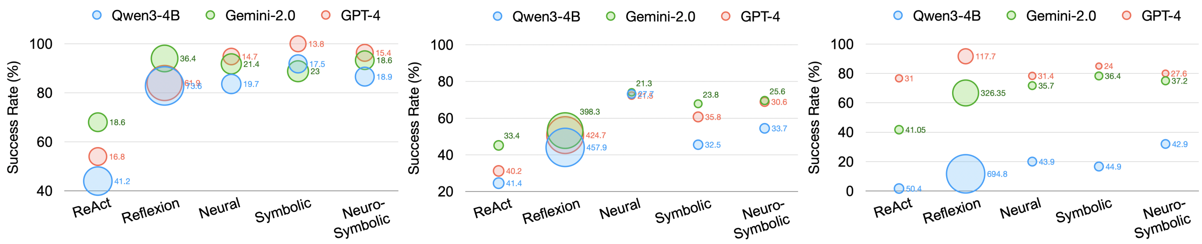
Impact of Types of Reflection. Table 4, presents the contributions of the three distinct types of reflections employed in our study. Our analysis reveals that no single type of reflection emerges as the definitive leader, suggesting that their combined implementation is integral to the robust performance exhibited by our OmniReflect framework. Notably, the data underscores the critical role of environment-level Error reflection, particularly in scenarios where task-level Progress reflections are absent, i.e. the ReAct agent is integrated with the OmniReflect Meta-advisor. Owing to the intricate structure of PDDL, where an action is successful only when all necessary conditions are met (as illustrated in TyreWorld, where the precondition for loosening requires that the agent possesses a wrench, the nut on the hub is tight, and the hub is grounded), Abstract reflections play a pivotal role in explaining these nuances that are often challenging to discern solely through error analysis.

For an in-depth discussion on the influence of calibration data size, as well as analysis and illustrative examples of constitutions produced by various models across multiple experiments, please consult Appendix section A.

4 Related Works

Constitutional AI (Bai et al., 2022) introduced the use of human-written constitutions to promote helpful and harmless behavior. In contrast, our frame-

[3]The average number of turns for Reflexion can be skewed due to a few failed examples. Nevertheless, the figure still provides an important comparison that OmniReflect is not impacted by outliers in the efficiency dimension.



(a) ALFWorld (b) BabyAI (c) PDDL

Figure 3: Comparison of efficiency and effectiveness across ReAct, Reflexion, and OmniReflect agents reveals that OmniReflect achieves success rates comparable to or exceeding those of Reflexion, while maintaining inference efficiency on par with ReAct. Bubble sizes denote the average number of interaction turns per task.

	ALFWorld	BabyAI	PDDL
OmniReflect-Neural	94.8	72.3	78.3
(-) Abstract	91.8	48.2	71.6
(-) Error	92.5	46.4	76.7
(-) Progress	88.8	54.5	75.0
ReAct + MetaAdvisor	96.3	64.2	80.0
(-) Abstract	92.5	61.6	73.3
(-) Error	90.6	52.7	75.0

Table 4: Success rate (%) on ALFWorld, BabyAI, and PDDL using OmniReflect-Neural and a ReAct agent with a meta-advisor, illustrating the contributions of individual reflection types. The largest performance drops are highlighted in red. GPT-4 is used as both the agent and meta-advisor in their respective settings.

work autonomously curates task-oriented constitutions focused on improving task-completion quality. Moreover, unlike their finetuning-based approach, we leverage prompt-based guidance.

Self-correction methods like Self-Consistency (Wang et al., 2023), Universal Self-Consistency (Chen et al., 2024a), and MCR (Yoran et al., 2023) enhance reasoning by aggregating or meta-reasoning over multiple CoT paths. Complementary work leverages iterative correction through natural language feedback (Madaan et al., 2024; Shinn et al., 2023), numeric rewards and meta-feedback (Pan et al., 2024), and introspective learning via Self-Play Fine-Tuning in weaker LLMs (Chen et al., 2024b). In contrast, OmniReflect performs reflection at both the environment and task level, using constitution-style rules. It enables robust, interpretable self-improvement in a single trial (without multiple reasoning chains or repeated sampling) while markedly boosting weaker LLMs without extra fine-tuning or inference overhead.

MemoryBank (Zhong et al., 2024), RET-LLM (Modarressi et al., 2023), and MemGPT (Packer et al., 2023) use structured memory or retrieval to persist knowledge, but face challenges like drift, size limits, and relevance filtering (Wu et al., 2024). In contrast, OmniReflect maintains a compact, coherent memory via periodic constitution summarization, avoiding unbounded growth.

Automatic prompt construction approaches like (Shin et al., 2020; Zhang et al., 2022; Xu et al., 2022; Prasad et al., 2022; Li and Liang, 2021; Pryzant et al., 2023; Guo et al.; Yang and Li, 2023; Tang et al., 2025) leverage LLMs as optimizers to adapt prompts for specific downstream tasks. In contrast, our approach uses a straightforward strategy by appending constitutions to system prompts that guide the model in using them effectively.

5 Conclusion

We introduced OmniReflect, a hierarchical reflection-driven framework that summarizes task and environment-level insights into reusable constitution, guiding LLM agents in complex environments. It operates effectively in Self-sustaining mode and Co-operative mode, where constitutions are derived from minimal calibration significantly boosting smaller agent's performance. Our Neural, Symbolic, and Neuro-Symbolic strategies balance adaptability with efficiency. Empirical results across ALFWorld, BabyAI, and PDDL demonstrate consistent improvements over strong baselines, underscoring OmniReflect's scalability, generalizability, and cost-efficiency in enhancing self-reflection, and adaptability in LLM agents, serving as a crucial benchmark toward building more efficient and autonomous language-based agents.

6 Limitations

While OmniReflect delivers strong performance gains, it introduces additional LLM calls, which may pose challenges for real-world deployment. However, we show that constitutions generated by smaller models (e.g., Qwen3-4B) can significantly improve the performance of larger models like GPT-4 and Gemini-2.0, suggesting that overhead can be mitigated through strategic model selection. Currently, constitutions are integrated without filtering, which may increase computational costs for models with limited context windows and introduce noise. Future work will explore more efficient constitution integration to reduce overhead and enhance usability. Though we evaluate OmniReflect in embodied agentic settings, extending it to broader reasoning and planning tasks remains a promising direction. Finally, while we use ReAct for its simplicity and minimal inference overhead, future efforts will explore combining OmniReflect with advanced strategies such as Self-Consistency to further strengthen agent robustness.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2024a. Universal self-consistency for large language models. In *ICML 2024 Workshop on In-Context Learning*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

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

Q Guo, R Wang, J Guo, B Li, K Song, X Tan, G Liu, J Bian, and Y Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. arxiv 2023. *arXiv preprint arXiv:2309.08532*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

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

Abhinav Jain, Chris Jermaine, and Vaibhav Unhelkar. 2024. Rag-modulo: Solving sequential tasks using experience, critics, and language models. *arXiv preprint arXiv:2409.12294*.

Dawei Li, Renliang Sun, Yue Huang, Ming Zhong, Bohan Jiang, Jiawei Han, Xiangliang Zhang, Wei Wang, and Huan Liu. 2025. Preference leakage: A contamination problem in llm-as-a-judge. *arXiv preprint arXiv:2502.01534*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2023. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*.

Yohei Nakajima. 2023. Babyagi. *Python. https://github.com/yoheinakajima/babyagi*.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2022. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Matthew Renze and Erhan Guven. 2024. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Tom Silver and Rohan Chitnis. 2020. Pddlgym: Gym environments from pddl problems. *arXiv preprint arXiv:2002.06432*.

Annalisa Szymanski, Noah Ziems, Heather A Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A Metoyer. 2025. Limitations of the llm-as-a-judge approach for evaluating llm outputs in expert knowledge tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, pages 952–966.

Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, Siyuan Lu, Yaliang Li, and Ji-Rong Wen. 2025. Unleashing the potential of large language models as prompt optimizers: Analogical analysis with gradient-based model optimizers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25264–25272.

Qwen Team. 2025. Qwen3. `https://github.com/QwenLM/Qwen3`.

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

Siye Wu, Jian Xie, Jiangjie Chen, Tinghui Zhu, Kai Zhang, and Yanghua Xiao. 2024. How easily do irrelevant inputs skew the responses of large language models? *arXiv preprint arXiv:2404.03302*.

Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Self-polish: Enhance reasoning in large language models via problem refinement. *arXiv preprint arXiv:2305.14497*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. 2022. Gps: Genetic prompt search for efficient few-shot learning. *arXiv preprint arXiv:2210.17041*.

Heng Yang and Ke Li. 2023. Instoptima: Evolutionary multi-objective instruction optimization via large language model-based instruction operators. *arXiv preprint arXiv:2310.17630*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5942–5966.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. 2022. Tempera: Test-time prompting via reinforcement learning. *arXiv preprint arXiv:2211.11890*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.

A Additional Results and Discussion

A.1 Amount of calibration data.

Table 5 shows that increasing the amount of calibration data generally improves the quality of the meta-advisor, thereby enhancing the downstream performance of ReAct agents that rely on it. However, performance gains begin to taper beyond a certain point. We exclude BabyAI and PDDL from this analysis, as using more than one example per task type would constitute nearly 50% of their respective test sets, undermining the goal of demonstrating that the meta-advisor can be calibrated with significantly fewer examples than required for evaluation. A calibration factor beyond 5 approaches this threshold for ALFWorld as well, and thus serves as the upper bound in our experiments.

ALFWorld - Success Rate			
Calibration Factor	Calibration Set Size	Qwen	GPT-4
1	6	76.9	96.3
3	18	79.1	97.8
5	30	**80.6**	**98.5**

Table 5: Effect of the calibration factor (number of examples per task type) on constitution quality and downstream ReAct agent performance, as measured on the ALFWorld dataset. GPT-4 is used as the meta-advisor.

A.2 Constitutions

This subsection presents representative examples spanning all three datasets and reflection types. On average, all models generate the highest number of abstract reflections (20–50 per dataset), while error reflections at the environment level are less frequent (typically fewer than 20). In contrast, the number of progress reflections scales with task complexity, as shown in the tables. GPT-4 consistently produced well-structured outputs, whereas Gemini 2.0 and Qwen3-4B encountered JSON-style formatting issues in over 50% of cases, necessitating complex post-processing to recover structured data.

An exception was observed with Gemini 2.0, which generated over 100 error reflections, diluting the effectiveness of targeted reflection and potentially contributing to its lower performance when guided by its own constitutions. GPT-4 produced the largest constitutions, often exhibiting high verbosity. While Gemini 2.0 and Qwen3-4B generated a comparable number of reflections, Qwen3-4B frequently yielded more coherent and concise summaries without sacrificing quality.

Notably, (without explicit guidance) Gemini 2.0 included priority annotations in its rules—for example: 'priority': 2, 'rule': 'Prioritize checking locations where target objects are most likely to be found (e.g., drawers, shelves, cabinets, countertop).', indicating an attempt to encode further structure within its reflective outputs that can be leveraged for reasoning.

Table 6, Table 7, and Table 8, show examples of different types of constitutions created by all three models. We have used majority voting to choose abstract and error constitution samples.

B ALFWorld

This section provides additional details and experimental results for the sequential decision making dataset ALFWorld. The embodied tasks are categorized into six types: Pick, Examine, Heat, Cool, Clean, and Pick Two. These tasks involve navigating a home environment to achieve specific goals, such as "*place the vase in the safe*" or "*inspect the book under the desk lamp*." Appendix Tables 9 provide a randomly chosen example annotation for one different types of tasks present in the dataset, along with a trajectory that solves the task.

C BabyAI

BabyAI environment was introduced in (Chevalier-Boisvert et al., 2018) and covers tasks to be performed in a grid environment. They can have multiple grid and minigrid sizes, ranging from a single minigrid to upto 9 minigrids. The minigrids can be of sizes 4∗4 to 7∗7. In this environment, the agent can see a 7∗7 grid in the direction it is currently facing. In most of the experiments, the agent is only exposed to this information which severely limits the global perspective of the complete grid

```
Abstract:
Use fridge for cooling
...
heat [object] with microwave [location] requires microwave to be closed
...
Plates can be found on countertops

Error:
{
    "mistake": "Went to locations that are not present in the environment.",
    "solution": "Carefully check the available locations before moving"
}
...
Progress:
[
You have located an apple,
...
You have reached the microwave,
...
]
```

Table 6: ALFWorld Constitution Examples

```
Abstract:
If you encounter a barrier while moving forward, turn left or right to explore a
different direction.
...
If you encounter a closed door, use the 'toggle and go through' command to open it
and proceed.
...
If you see multiple doors, prioritize the closest one first.
...
If you see an object, note its color and position for future reference.

Error:
{
    "mistake": "Attempted to move forward into a barrier",
    "solution": "Should have turned right first to explore the room further"
}
...
{
    "mistake": "Attempted to open the door with an unrecognized action",
    "solution": "Should have checked valid actions before attempting to open the
    door"
}

Progress:
[
You have found a blue key, now find a blue door.
...
]
```

Table 7: BabyAI Constitution Examples

```
Abstract:

Gripper Example
If both grippers are occupied, move to the target room to drop the objects.
...
Blockworld Example
If the robot arm is holding a block, it can put down the block or stack it on
another clear block.
...
Barman Example
If you need to transfer an ingredient from a shot glass to a shaker, ensure the
shaker is clean and at the appropriate level.
...
Tyreworld Example
Complete the process on one hub before moving to the next, including jacking down
the hub after replacing the wheel and tightening the nuts.

Error:

{
        "mistake": "Attempted to shake a cocktail without all ingredients in the
        shaker",
        "solution": "Ensure all required ingredients are in the shaker before
        shaking"
}
      ...
{
        "mistake": "Inefficient sequence of actions",
        "solution": "Plan the sequence of actions to minimize the number of steps,
        such as filling all ingredients in the shot glass before transferring to the
         shaker"
}

Progress:

[
  "You have moved to roomb with ball1 and ball2, now you should drop ball1 and ball2
   in roomb.",
  "After dropping ball1 and ball2, you should move back to rooma to pick up ball3
  and ball4.",
  "Once you have picked up ball3 and ball4, move to roomb and drop them there.",
  "After dropping ball3 and ball4, return to rooma to pick up ball5 and ball6.",
  "Finally, move to roomb and drop ball5 and ball6 to complete the task."
]
...
[
  "You have been repeatedly attempting to unstack b5 from b3, which is not a valid
  action. Instead, consider other valid actions.",
  "Since b5 is clear and the robot arm is empty, you should pick up b5.",
  "After picking up b5, you can put it down on the table to free up b3.",
  "Once b3 is clear, you can unstack b3 from b4.",
  "After unstacking b3 from b4, you can put b3 on the table to free up b4.",
  "Then, you can unstack b4 from b2.",
  "After unstacking b4 from b2, you can put b4 on the table to free up b2.",
  "Next, you can unstack b2 from b1.",
  "After unstacking b2 from b1, you can put b2 on the table to free up b1.",
  "Now, you can stack b1 on b2 to satisfy the first goal condition.",
  "Then, you can pick up b4 and stack it on b1 to satisfy the third goal condition
  .",
  "Finally, you can pick up b3 and stack it on b5 to satisfy the second goal
  condition."
]
...
```

Table 8: PDDL Constitution Examples

Table 9: ALFWorld: PUT Task

```
Task: Find some apple and put it in sidetable.
Initial observation: You are in the middle of a room. Looking quickly around you,
you see a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a
countertop 1, a diningtable 3, a diningtable 2, a diningtable 1, a drawer 1, a
fridge 1, a garbagecan 1, a microwave 1, a sidetable 1, a sinkbasin 1, a stoveburner
 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1. \newline
Your task is to: find some apple and put it in sidetable.
Action 1: think: To solve the task, I need to find and take an apple, then put it in
 sidetable.
Observation 1: OK.
Action 2: think: First I need to find an apple. An apple is more likely to appear in
 fridges (1), diningtables (1-3), sidetables (1), countertops (1), sinkbasins (1),
garbagecan (1). I can check one by one, starting with fridge 1.
Observation 2: OK.
Action 3: go to  fridge 1.
Observation 3: The fridge 1 is closed.
Action 4: open fridge 1.
Observation 4: You open the fridge 1. The fridge 1 is open. In it, you see a lettuce
 2, a mug 2, a potato 2, and a tomato 1.
Exploring: ...
Action 11: go to  garbagecan 1.
Observation 11: On the garbagecan 1, you see a apple 3, and a egg 3.
Action 12: think: Now I find an apple (3). Next, I need to take it.
Observation 12: OK.
Action 13: take apple 3 from garbagecan 1.
Observation 13: You pick up the apple 3 from the garbagecan 1.
Action 14: think: Now I find an apple (3). Next, I need to put it in/on sidetable 1.
Observation 14: OK.
Action 15: go to  sidetable 1.
Observation 15: On the sidetable 1, you see a cup 1, a lettuce 1, a peppershaker 3,
a potato 1, and a saltshaker 1.
Action 16: put apple 3 in/on sidetable 1.
Observation 16: You put the apple 3 in/on the sidetable 1.
```

which leads to lot of blind exploration. Figure 4 provides an example of grid structure and objects used in BabyAI environment.

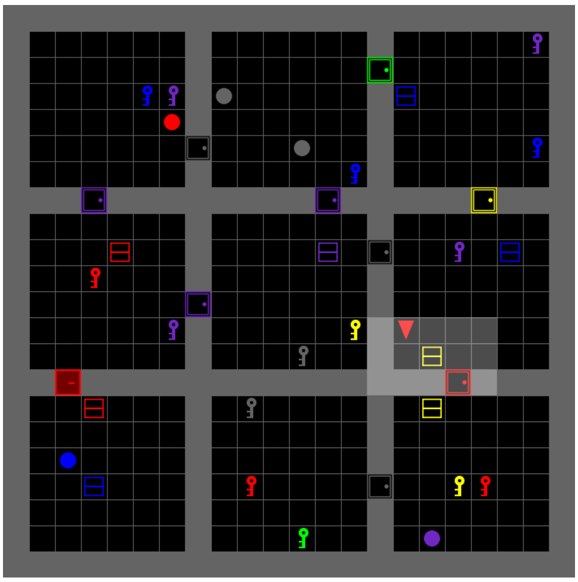


Figure 4: Visualization of a BabyAI grid environment showcasing balls, boxes, keys, and doors, with the red triangle marking the agent's location and orientation.

D PDDL

PDDL benchmark was made accesible using (Silver and Chitnis, 2020). It contains four distinct environments that are used for this work: Gripper, Blockworld, Barman, and Tyreworld. Gripper and Blockworld provide an initial state and a goal state without explicit task instructions. However Barman and Tyreworld provide explicit task goals. The agent is expected to reason, plan, and navigate to achieve the goal state. Examples of each dataset can be found in the Table 10 below.

E Prompts

All prompts used in our experiments are outlined in this subsection.

E.1 System Prompts

System prompts are specific to the environment which outline action templates, and high level guidelines for solving tasks in the environment. Table 11 shows an example for ALFWorld, Table 12 shares an example for BabyAI, and finally Table 13 presents system prompt for PDDL.

Dataset	Example Task
Gripper	The goal is to satisfy the following conditions: ball1 is at roomb. ball2 is at roomb. ball3 is at roomb. ball4 is at roomb.
Blockworld	The goal is to satisfy the following conditions: b1 is on b2. b2 is on b6. b3 is on b7. b5 is on b3. b6 is on b5. b7 is on b4.
Barman	The goal is to satisfy the following conditions: shot1 contains cocktail2. shot2 contains cocktail3.
Tyreworld	The goal is to satisfy the following conditions: Wheel r1 is inflated. r2 is on the-hub2. w1 is in boot.

Table 10: Examples of sample tasks from datasets comprised in PDDL

Table 11: ALFWorld ReAct Prompt

```
Interact with a household to solve a task.
You need to generate actions that strictly follow the below templates:
1. goto [location]
2. take [object] from [location] put [object] in/on [location]
3. open [something]
4. close [something]
5. toggle [object][location]
6. clean [object] with [something]
7. heat [object] with [receptacle]
8. cool [object] with [receptacle]

Here are two examples. They are very relevant. Please use the actions in these
examples as your guidelines.
\textit{Example 1: Truncated}
\textit{Example 2: Truncated}
```

Table 12: BabyAI ReAct Prompt

```
You are placed in a room and you need to accomplish the given goal with actions.

You can use the following actions:

- turn right

- turn left

- move forward

- go to <obj> <id>

- pick up <obj> <id>

- go through <door> <id>: <door> must be an open door.

- toggle and go through <door> <id>: <door> can be a closed door or a locked door.
If you want to open a locked door, you need to carry a key that is of the same color
 as the locked door.

- toggle: there is a closed or locked door right in front of you and you can toggle
it.


\textit{Example 1: Truncated}
\textit{Example 2: Truncated}
```

Table 13: PDDL ReAct Prompt

```
---------blockworld---------
The robot has four actions: pickup, putdown, stack, and unstack. The domain assumes
a world where there are a set of blocks that can be stacked on top of each other, an
 arm that can hold one block at a time, and a table where blocks can be placed.
    The actions defined in this domain include:
    pickup <block>: pick up a clear block
    putdown <block>: put down a block on the table
    stack <block> <block>: stack a block on top of another block.
    unstack <block> <block>: unstack a block from on top of another block
---------barman---------
You are a robot barman that manipulates drink dispensers, shot glasses and a shaker.
 You have two hands. The goal is to find a plan that serves a desired set of drinks.
 Here are the actions you can do. Each valid action is a short phrase following
fixed patterns:

    <hand> grasp <container>: Grasp a container
    <hand> leave <container>: Leave a container on the table
    fill-shot <shot> <ingredient> <hand1> <hand2> <dispenser>: Fill a shot glass
    with an ingredient from dispenser
    refill-shot <shot> <ingredient> <hand1> <hand2> <dispenser>: Refill a shot glass
     with an ingredient from dispenser
    empty-shot <hand> <shot> <beverage>: Empty a shot glass
    clean-shot <shot> <beverage> <hand1> <hand2>: Clean a shot glass
    pour-shot-to-clean-shaker <shot> <ingredient> <shaker> <hand1> <level1> <level2
    >: Pour an ingredient from a shot glass to a clean shaker from level1 to level2
    pour-shot-to-used-shaker <shot> <ingredient> <shaker> <hand1> <level1> <level2>:
     Pour an ingredient from a shot glass to a used shaker from level1 to level2
    empty-shaker <hand> <shaker> <cocktail> <level1> <level2>: Empty a shaker
    containing cocktail from level1 to level2
    clean-shaker <hand1> <hand2> <shaker>: Clean a shaker
    shake <cocktail> <ingredient1> <ingredient2> <shaker> <hand1> <hand2>: Shake a
    cocktail in a shaker
    pour-shaker-to-shot <beverage> <shot> <hand> <shaker> <level1> <level2>: Pour a
    beverage from a shaker to a shot glass from level1 to level2
---------gripper---------
You are a robot with a gripper that can move objects between different rooms. Your
name is Robby.
    There are three actions defined in this domain:
    move <room1> <room2>: This action allows the robot to move from one room to
    another.
    pick <obj> <room> <gripper>: This action allows the robot to pick up an object
    using the gripper.
    drop <obj> <room> <gripper>: This action allows the robot to drop an object that
     it is carrying.
---------tyreworld---------
Your goal is to replace flat tyres with intact tyres on the hubs. Remember to open
boot first to get tools you need. Intact tyres should be inflated. The nuts should
be tight on the hubs. The flat tyres, wrench, jack, and pump should be in the boot.
The boot should be closed.
    There are 13 actions defined in this domain:
    open <container>
    close <container>
    fetch <object> <container>
    put-away <object> <container>
    tighten <nut> <hub>
    jack-up <hub>
    jack-down <hub>
    undo <nut> <hub>
    do-up <nut> <hub>
    remove-wheel <wheel> <hub>
    put-on-wheel <wheel> <hub>
    inflate <wheel>
\textit{Example 1: Truncated}
\textit{Example 2: Truncated}
```

E.2 Symbolic Prompts

Table 14, Table 15, Table 16 provide symbolic system prompts used by our OmniReflect-Symbolic System to provide environment level guidance.

E.3 Reflection Prompts

Table 17 present reflection prompts used for generating abstract, error, and progress level reflections for ALFWorld, BabyAI, and PDDL datasets. They share similar content, with the exception of examples used to demonstrate reflection examples. In Neuro-Symbolic case, we use templated responses that are generated by OmniReflect-Symbolic on the calibration set, as few shot examples. Table 18 provides the simple prompt used for summarization across datasets, and for all models.

F Symbolic Reflections

We use engineered prompts and regular expressions for generating symbolic reflections. All prompts have been presented in Section E. The regular expressions used for progress tracking in OmniReflect-Symbolic system are presented below.
ALFWorld. Example of ALFWorld tracking and reflection regular expressions we used can be found in Table 19.
BabyAI. We provide reflection for these situations:

- You are going in a circle, just turn right or turn left and move forward and check valid actions.
- You have found a key, use it to open a same colored locked door in the path if needed. DO NOT DROP the key before you unlock the necessary doors.
- I have found the door, If my task is to unlock a door, I will unlock the door, else I will toggle and go through the door. template for this action is toggle and go through <color> [closed|locked] <door> <id>
- I found the target object. I will move towards it and pick it up if needed.
- Now you should drop this in a free location that does not block the path for my next steps. You cannot carry two items, so you MUST drop this before picking up the next item. DO NOT DROP if you holding a KEY. KEY should be used to unlock the door and then you can drop it.

PDDL. Apart from sharing the left-over subgoals, heuristic reflection takes a simple form of 'You are doing: ' and 'you should be doing: ' where information is populated under the following situations:

- Gripper
  - When objects that are at the destination are being accidentally picked up again
  - When the agent is performing sub-optimal pick ups
  - When the item incorrectly drops an object
- Blockworld
  - When agent attempts to use incorrect commands, such as pick up for unstack
  - When the agent is performing sub-optimal stacking, i.e. if the goal state is 1,2,3,4, it attempts to perform 1,2 and 3,4 separately. Since stacks cannot be stacked on top of each other, we warn the agent when it enters this situation
  - When the agent is stuck in a loop of stacking and unstacking incorrect blocks
- Barman
  - When agent does not leave objects in between tasks, which leads to incorrect grasping
  - When agent uses unclean objects, and does not perform cleaning, for instance, pour-shot-to-clean-shaker does not do anything if a clean shaker is unavailable
  - When the agent incorrectly assumes task is done, due to incorrect order of ingredient mixing
- Tyreworld
  - When agent retrieves unnecessary tools
  - When agent forgets steps required for preparation such as removing nuts or jacking a wheel

Table 14: ALFWorld Symbolic System Prompt

```
You need to go to a location or an object before using it or placing the objects at
the location.
For example you need to `go to garbagecan 1` or `go to microwave 1` before using or
placing the objects at the `garbagecan 1` or `microwave 1

You can only pick up or hold one object at a time.

Everything in the environment is labelled with a numbers. You ALWAYS need to use the
 number that follows when referring to anything in the environment.
Valid example:`take lettuce 1 from countertop 1`
Invalid example: `take lettuce from countertop 1

You MUST Alternate between Thinking and Action generation. An example of think is `
think: CD can be found on desk.` and An example of action is `take cd 1 from desk
1.`

You can ONLY use microwaves for heating. Once you are at a microwave, you can
directly try to heat the item.
For example: For the action `go to microwave 1' can directly be followed by the
action `heat apple 1 with microwave 1'

Once you are at a fridge, you can directly try to cool the item.
For example: For the action `go to fridge 1' can directly be followed by the action
`cool lettuce 1 with fridge 1'

For tasks involving look or examine using desklamp you need to find a desklamp.
Once you are at a location with desklamp you can directly use the desklamp. The
correct usage is through action of `use desklamp 1' for using desklamp 1.

For clean or cleaning tasks first obtain the item to be cleaned. You need to then
clean the item at sinkbasin.
Once you are at a sinkbasin, you can directly try to clean the item.
For example: For the action `go to sinkbasin 1' can directly be followed by the
action `clean plate 1 with sinkbasin 1'
```

Table 15: BabyAI Symbolic System Prompt

```
You are in a grid environment with multiple multiple minigrids.
Doors connect different mini grids that are separated by walls. You should go
through doors if necessary to get to the destination.
For instance, if you are at row 4 and column 3 , facing up, and your target is at
row 15 column 1, you should find a path to go down to row 15 and left to column 1 by
 toggling doors in between as needed.
YOU NEED KEY ONLY IF THE DOOR IS LOCKED. If a door is locked then you should find
the same colored key to unlock and go through the door.
You only need a key once to toggle through the door. In the next turns, the door is
no longer locked, do you do not need to pick up that color keys unnecessarily.

You can ONLY hold only one object at a time. If you are able to pickup an object,
then drop what you are currently holding and then pickup the new object.

If you are facing a wall, turn left or turn right to explore other objects
If you to navigate to an object behind you, you can turn back. For example, If you
are in minigrid 0, with direction ^ then turn back to access rest of the grid.

DO NOT repeat the turn multiple times, because you will get lost.

If you are blocked or having trouble picking up an object, you MUST turn to an empty
 cell and drop what you are currently holding, you CANNOT drop at the same location,
 as you are facing an object and then pickup the blocking object and move it out of
the way.
Once path is clear, you move or try to pick up the object that is blocking you.
You can ONLY drop objects in empty spaces. DO NOT DROP keys before you use them on
the same colored door. You should drop them after toggling through the door
You MUST NOT drop an object immediately, as that would mean you are dropping it in
the same place. So you MUST turn to an empty spot and then drop it. DO NOT DROP it
in a cell that blocks your path to the next step.

-----------------------

First Turn: You should first generate a thought with a path from your minigrid to
the destination minigrid with all the doors you need to go through. First determine,
 which door you should use to exit your grid if needed. For example, To go from
minigrid 0 to minigrid 5, I need to go through yellow closed door 1 .....\n Generate
 this in less than 6 lines.
```

Table 16: PDDL Symbolic System Prompt

```
These are just guidelines and not the complete commands, so you should generate a
correct command in the correct template.
If your subgoal is that a shot contains an ingredient, you should do the following
steps:
    1. grasp the correct shot
    2. fill-shot using the dispenser that contains the ingredient

If your subgoal is that a shot contains a cocktail, you should do the following
steps:
    Phase 1: Collecting all ingredients into a shaker, for each ingredient in the
    cocktail do the following
        1. grasp the correct shot
        2. fill-shot using the dispenser that contains the ingredient
        3. pour-shot-to-clean-shaker
        4. clean-shot
    Phase 2: Shake and serve
        1. leave the shot
        2. grap the shaker with all the ingredients
        3. shake
        4. pour-shaker-to-shot

Here is an example of making a cocktail with ingredient 2 and ingredient 1 in shot3:
-> Filling ingredient 2
left grasp shot3
fill-shot shot3 ingredient2 left right dispenser2
pour-shot-to-clean-shaker shot3 ingredient2 shaker1 left l0 l1
clean-shot glass shot3 with ingredient2 with hand left holding shot glass and right

-> Filling ingredient 1
fill-shot shot3 ingredient1 left right dispenser1
pour-shot-to-used-shaker shot3 ingredient1 shaker1 left l1 l2
clean-shot glass shot3 with ingredient1 with hand left holding shot glass and right

-> Shake and serve
left leave shot3
right grasp shaker1
shake cocktail3 ingredient2 ingredient1 shaker1 right left
pour-shaker-to-shot cocktail3 shot3 right shaker1 l2 l1
Here is subgoal guidance for your current task, they are NOT EXACT commands, they
are just guidance:
After you complete a subgoal, leave any objects you are holding.
```

Table 17: Reflection Prompts

```
>> Abstract
Generate a constitution specific for solving a {tasktype} task and about the
environment.
The constitution should be solely based on the observation in this environment, and
should not contain general rules about regular world.
The rules in the constitution should be generalizable, abstract, correct, and
profound.
Some examples could include: Use microwave for heating or Tomatoes can be found in
fridge, among others. <- ALFWorld
Some examples could include: If you are facing a wall, turn around and continue
exploration. <- BabyAI
Some examples could include: If you have only one arm, you cannot pick up two items
<- PDDL
The constitution should be in a python list format (enclosed in [])

>> Error
Generate a constitution specific for solving this task covering the potential
mistakes performed so far and your suggestions on how to fix it.
The constitution should be solely based on the observation in this environment, and
should not contain general rules about regular world.
The constitution should be in a python list of dictionaries format without any extra
 text in a single line.
You should thoroughly analyze the current trajectory and only provide feedback if a
mistake happened so far. Sometimes mistakes can be indicated by the observation `
Nothing happens`.
DO NOT predict future mistakes, or share advice about future steps.
If there are no mistakes so far, then return an empty list
If efficiency of the trajectory can be improved, you should add that as well.
Here is an example: [{'mistake': 'Cabinet was not opened', 'solution': 'Open the
cabinet next time}, ...] <- ALFWorld
Here is an example: [{'mistake': 'Going in circles', 'solution': 'Stop turning same
way and going in circles...}, ...] <- BabyAI
Here is an example: [{'mistake': 'Attempted to pick up a block that is stacked', '
solution': 'Should use unstack...}, ...] <- PDDL
>> Progress

Critically examine the trajectory so far to solve the task, and generate explicit
feedback for solving leftover subtasks.
Example: For a task of placing a heated apple in a garbage, one feedback example
could be `You have heated the apple, now you should pick it up and go to garbagecan`
 <- ALFWorld
Example: For a task of going through a green door, one feedback example could be `
You have located a green key, now pick it up and locate a green door.` <- BabyAI
Example: An example could be: I have poured ingredient 1 into the shaker. I should
then shake and serve in a clean shot class. <- PDDL
The constitution should be in a python list format (enclosed in []) without any
extra text in a single line.
```

Table 18: Summarization Prompt

```
Inspect and summarize the constitution you have build over time by exploring the
environment and solving numerous tasks.
The resulting summary should be usable by any other agent to quickly solve tasks by
using the knowledge built using your experience. <- Abstract
The resulting summary should be usable by any other agent to avoid making any
mistakes that were made. <- Error
There should not be duplicates in the constitution. You should be clear and concise
while summarizing.
You can create new rules by summarizing multiple rules together without losing
information.
Here is the current constitution: [...]
The summarized constitution should be in a python list format (enclosed in []).
```

Table 19: Progress Tracking Regular Expressions Examples

```
Type: examine
Goal: look at bowl under the desklamp.
Patterns:
^(?=.* you see)(?=.*a bowl \d+)
You pick up the bowl \d+
^(?=.* you see)(?=.*a desklamp)
--------

Type: puttwo
Goal: put two soapbar in garbagecan.
Patterns:
^(?=.* you see)(?=.*a soapbar \d+)
You pick up the soapbar \d+
You put the soapbar \d+ in/on the garbagecan \d+
^(?=.* you see)(?=.*a soapbar \d+)
You pick up the soapbar \d+
You put the soapbar \d+ in/on the garbagecan \d+
--------

Type: cool
Goal: put a cool tomato in microwave.
Patterns:
^(?=.* you see)(?=.*a tomato \d+)
You pick up the tomato \d+
You cool the tomato \d+ using the
You put the tomato \d+ in/on the microwave \d+
--------
```