EXPERIENTIAL REFLECTIVE LEARNING FOR SELF-IMPROVING LLM AGENTS

Marc-Antoine Allard* **Arnaud Teinturier*** **Victor Xing*** **Gautier Viaud**
Illuin Technology
{marc-antoine.allard,victor.xing}@illuin.tech

ABSTRACT

Recent advances in large language models (LLMs) have enabled the development of autonomous agents capable of complex reasoning and multi-step problem solving. However, these agents struggle to adapt to specialized environments and do not leverage past interactions, approaching each new task from scratch regardless of their accumulated experience. We introduce **Experiential Reflective Learning** (ERL), a simple self-improvement framework that enables rapid environment adaptation through experiential learning. ERL reflects on task trajectories and outcomes to generate heuristics, capturing actionable lessons that transfer across tasks. At test time, relevant heuristics are retrieved based on the current task and injected into the agent's context to guide execution. On the Gaia2 benchmark, ERL improves success rate by 7.8% over a ReAct baseline, with large gains in task completion reliability, and outperforms prior experiential learning methods. Through systematic ablations, we find that selective retrieval is essential and that heuristics provide more transferable abstractions than few-shot trajectory prompting. These results demonstrate that reflecting on single-attempt experiences to extract transferable heuristics enables effective agent self-improvement.

1 INTRODUCTION

Agentic systems powered by large language models (LLMs) are increasingly deployed for complex tasks requiring multi-step planning, reasoning, and tool use (Luo et al., 2025; Yao et al., 2025; Wei et al., 2025). However, general-purpose agents often fail to adapt to new environments with unfamiliar tools and domain-specific conventions (Chen et al., 2026). Fine-tuning can enable such adaptation but is resource-intensive, infeasible for closed-source models, and does not support continuous learning. These limitations have motivated research into experiential memory systems that enable parameter-free improvement through accumulated experience (Hu et al., 2026).

Recent work has explored how to make LLM agents learn from accumulated experience without parameter updates (Cai et al., 2025; Fang et al., 2026; Sarukkai et al., 2025; Ouyang et al., 2025; Fu et al., 2024; Zhao et al., 2024). ExpeL (Zhao et al., 2024) extracts reusable insights by comparing successful and failed trajectories, using Reflexion to retry each task until success. Extracted insights are then concatenated into every test prompt regardless of task relevance, an approach that scales poorly as experience accumulates. AutoGuide (Fu et al., 2024) creates context-aware guidelines on offline training tasks by contrasting paired trajectories with different outcomes. At test time, it performs context identification and guideline retrieval at every agent turn, incurring substantial overhead, and provides no guidance when the current state fails to match any stored context. Both methods require multiple rollouts per task to construct contrastive trajectory pairs, an assumption that breaks down in practical agent deployment where tasks cannot be retried.

In this work, we propose **Experiential Reflective Learning** (ERL), an experiential memory framework designed for efficient self-improvement in new environments. ERL builds a pool of reusable heuristics that capture effective strategies and failure modes, generated by reflecting on past experience trajectories and their outcomes. Then, for each new task, an LLM scores stored heuristics for relevance and injects the top candidates into the agent's context, providing task-specific guid-

*Equal contribution.

ance. ERL extracts heuristics from single-attempt trajectories, enabling efficient adaptation without curated training sets or repeated execution. Moreover, ERL heuristics preserve granular trajectory details that existing methods lose through cross-task aggregation.

On the Search and Execution splits of Gaia2 (Froger et al., 2025), ERL achieves an overall success rate of 56.1%, an improvement of +7.8% over a ReAct baseline and outperforming ExpeL and AutoGuide. We carry out experiments highlighting that heuristic retrieval plays a critical role in improving performance and reliability, lessons from failures and successes benefit different task types, and heuristics provide more transferable abstractions than raw agent trajectories.

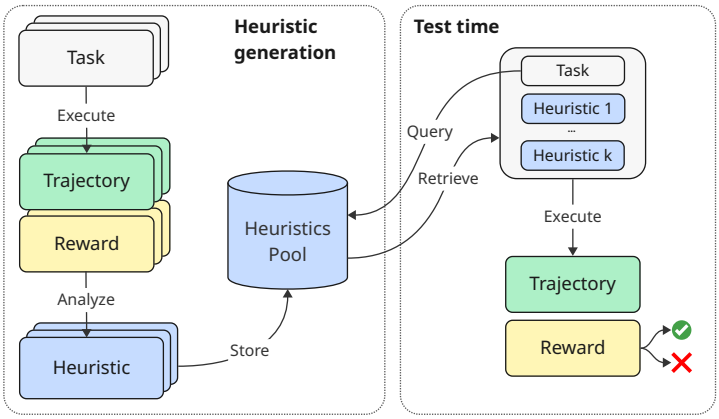


Figure 1: The ERL framework. During experience accumulation (left), the agent reflects on task outcomes to generate heuristics stored in a persistent pool. At test time (right), relevant heuristics are retrieved and injected into context for new tasks.

2 METHODS : EXPERIENTIAL REFLECTIVE LEARNING

ERL consists of two components (Figure 1): heuristic generation from task experience, and retrieval-augmented execution for new tasks.

Heuristic generation. We consider an agent operating in an environment that provides binary success/failure feedback upon task completion (we analyze performance when outcome signals are unavailable in Appendix C). As the agent executes tasks, it accumulates experiences consisting of the task description, the execution trajectory (reasoning steps, tool calls, and outputs), and the outcome signal. After each task, the agent reflects on this experience to generate a structured heuristic containing: (1) an analysis identifying what led to success or failure, and (2) a learned guideline with explicit trigger conditions and recommended actions (e.g., "When sending emails to calendar attendees, first resolve names to email addresses via the Contacts tool before calling the email API"). These heuristics are stored in a persistent pool.

Retrieval-augmented execution. When facing a new task, the agent retrieves relevant heuristics from the pool to guide its execution and improve from past experiences. An LLM analyzes the new task, decomposes it into potential sub-tasks and action steps, and scores each stored heuristic for relevance. Selection is made based on the similarity between the stored and current task descriptions, the diversity of experiences to cover a range of potential lessons, and the informativeness of the guideline content. The top-k heuristics are then injected into the agent's system prompt, ensuring that the agent receives task-specific advice rather than being overwhelmed with the full heuristic pool. Appendix F provides an example heuristic and a trajectory where the agent explicitly references the guidance it was given, and Appendix G contains the prompts for heuristic generation and retrieval.

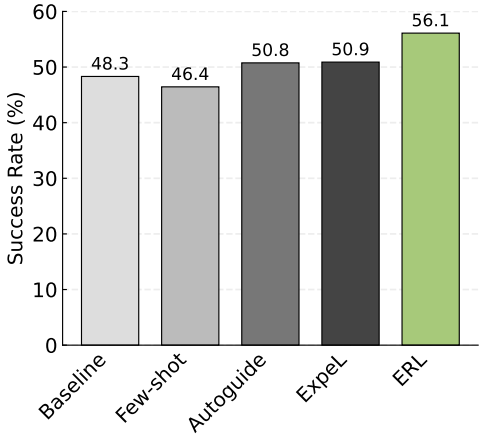
3 EXPERIMENTS

Experimental setup. We evaluate ERL against existing approaches on Gaia2 (Froger et al., 2025), a recent benchmark that assesses long-horizon agentic capabilities within a simulated mobile environment containing 12 applications and 101 tools. We focus on the Search and Execution splits which contain tasks involving information retrieval and multi-step execution capabilities.

We use the default ReAct agent scaffold of Gaia2 as the baseline method in our evaluations. ERL augments this baseline by injecting retrieved heuristics into the agent's system prompt before execution, requiring no modifications to the core ReAct loop. Unless otherwise noted, we evaluate ERL with $k = 20$ heuristics retrieved by an LLM (see Appendix C.2 for discussions on the retrieval method and choice of k). We use GPT-5-mini as the agent backbone for all methods.

A key feature of Gaia2 is its organization into *universes*: isolated data partitions that expose identical applications and tools but contain completely disjoint information (e.g., different contacts, files, and calendar events). This structure enables evaluation without knowledge contamination: we accumulate heuristics on 8 universes (112 execution / 132 search tasks) and evaluate on 2 held-out test universes (48 / 28 tasks). This setup mirrors real-world deployment where agents typically operate within a fixed set of tools while encountering continuously evolving data. Additional evaluation details are available in Appendix B.

Main results. Figure 2 summarizes performance across methods. ERL achieves a 56.1% overall success rate, an improvement of +7.8% over the ReAct baseline and +5.2% over the strongest prior method (ExpeL at 50.9%). Gains are consistent across task types: +8.3% on Execution and +7.1% on Search relative to the baseline. Prior methods exhibit uneven performance across splits, as AutoGuide achieves strong Search results (61.9%) but underperforms on Execution (39.6%, below baseline), while ExpeL shows modest improvements on both. Few-shot prompting with raw trajectory demonstrations also fails to improve over the baseline (46.4%).



(a) Averaged across Search and Execution splits.

Method	Success Rate (%)		
	Exec.	Search	Overall
Baseline	43.1	53.6	48.3
Few-shot	41.7	51.2	46.4
ExpeL	45.8	56.0	50.9
AutoGuide	39.6	61.9	50.8
ERL (Ours)	<u>51.4</u>	60.7	<u>56.1</u>
Ablations:			
no retrieval	44.4	<u>63.1</u>	53.8
embedding retrieval	50.7	56.0	53.3
only failures	50.0	**67.9**	**58.9**
only successes	**52.1**	47.6	49.9

(b) Per-split results and ablations.

Figure 2: Success rates on the Gaia2 test universes (3 runs). **Bold**: best; <u>underline</u>: second best.

ERL improves agent reliability. Figure 3 compares pass@3 and passˆ3 between the baseline and ERL. pass@3 measures the fraction of scenarios where the agent succeeds at least once across three runs, capturing whether a task is within the agent's capability. passˆ3 requires success on all three runs, measuring whether the agent can reliably complete a task (Chen et al., 2021; Yao et al., 2025). ERL yields substantial gains on passˆ3 (+8.3% in Execution and +10.6% in Search), indicating more stable performance across runs. Improvements on pass@3 are comparatively smaller, suggesting only marginal gains in newly solved scenarios.

Heuristics generalize better than trajectories. A simple approach to in-context learning from experiences is to append raw trajectories as few-shot demonstrations. However, this approach fails

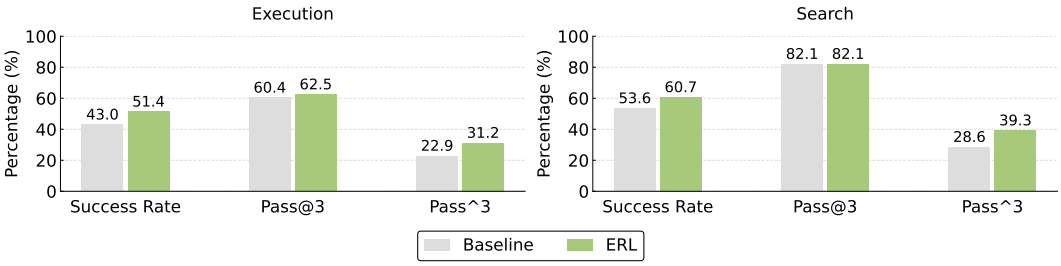


Figure 3: Success rate, pass@3 and pass^3 comparison between baseline and ERL

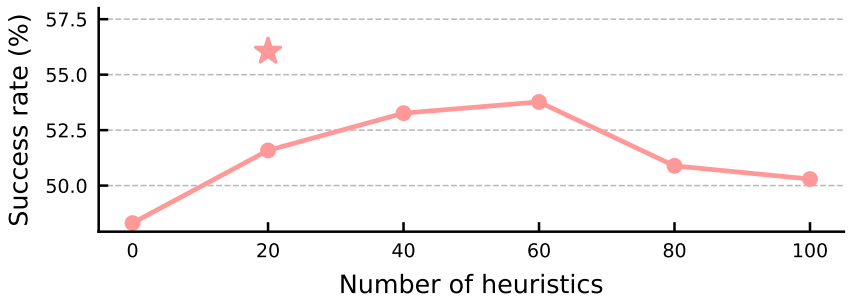


Figure 4: Success rate vs. number of randomly selected heuristics. ERL with LLM-based retrieval of $k = 20$ heuristics (★) outperforms all random configurations.

to improve performance (-1.9% vs. baseline), suggesting that raw trajectories lack the actionable insights necessary for the agent to improve on new scenarios. On the other hand, heuristics provide distilled strategic principles that generalize across tasks and are more context-efficient. We show that this advantage holds across token budgets, as heuristics consistently outperform trajectories even when controlling for context length (Appendix C).

Retrieval quality matters more than quantity. ERL retrieves task-relevant heuristics using an LLM-based selection mechanism. To validate the value of this choice, we compare against random selection. Figure 4 shows average success rate as a function of the number of randomly selected heuristics. Performance exhibits a non-monotonic pattern, peaking around 40–60 heuristics before degrading with excessive inclusion. This motivates the need for a selection mechanism that identifies the most relevant heuristics from an extensive pool, rather than relying on quantity alone. LLM-based retrieval with $k = 20$ achieves 56.1% overall, outperforming both embedding-based retrieval using Qwen3-Embedding-0.6B (Zhang et al., 2025) (53.3%, Table 2b) and the best random selection configuration (53.8%), confirming that retrieval quality matters more than heuristic quantity.

Failure heuristics favor Search; success heuristics favor Execution. Prior methods explore learning specifically from past failures or successes (Sarukkai et al., 2025; Zhao et al., 2024; Shinn et al., 2023). We examine whether the outcome of source experiences affects heuristic quality. Table 2b shows that failure-derived heuristics substantially outperform success-derived ones overall. The effect is split-dependent (Table 2b): failure heuristics excel on Search (+14.3% over baseline) by providing negative constraints that prune ineffective strategies, whereas success heuristics work best on Execution (+9.0%) by reinforcing proven action sequences. While failure-only retrieval yields the highest overall score (58.9%), this reflects strong Search performance at the expense of Execution tasks. In practical deployment scenarios where the task distribution is unknown, retrieving from both sources offers a reliable compromise.

4 Conclusion

We introduce Experiential Reflective Learning (ERL), a framework enabling LLM agents to improve through accumulated experience. By distilling trajectories into reusable heuristics and retrieving

relevant guidance at test time, ERL outperforms prior experiential learning methods and improves task completion reliability on Gaia2. Our analysis reveals that (1) heuristics transfer better than raw trajectories, (2) LLM-based retrieval outperforms random and embedding-based selection, and (3) learning from failures versus successes impacts different task types differently. Future work could bootstrap heuristic accumulation via synthetic task generation or address challenges in scaling heuristic pools, such as resolving conflicting guidelines and maintaining retrieval quality.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jinsong Su, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. Flex: Continuous agent evolution via forward learning from experience, 2025. URL https://arxiv.org/abs/2511.06449.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Grounded test-time adaptation for llm agents, 2026. URL https://arxiv.org/abs/2511.04847.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2026. URL https://arxiv.org/abs/2508.06433.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Are: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37: 119919–119948, 2024.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026. URL https://arxiv.org/abs/2512.13564.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. Mcp-universe: Benchmarking large language models with real-world model context protocol servers, 2025. URL https://arxiv.org/abs/2508.14704.

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, 2025. URL https://arxiv.org/abs/2509.25140.

Vishnu Sarukkai, Zhiqiang Xie, and Kayvon Fatahalian. Self-generated in-context examples improve llm agents for sequential decision-making tasks, 2025. URL https://arxiv.org/abs/2505.00234.

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

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025. URL https://arxiv.org/abs/2506.05176.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A τ^2-BENCH EVALUATION

While Gaia2 evaluates agents in a single-control setting, real-world deployment often requires agents to coordinate with users who actively participate in task resolution. We therefore evaluate ERL on τ^2-bench (Barres et al., 2025), a benchmark spanning three customer service domains (Airline, Retail, and Telecom).

Table 1 shows that ERL improves overall success rate over the baseline (0.380 vs. 0.367), with gains on Airline and Retail but a slight drop on Telecom. On Airline and Retail, ERL improves passˆ3 while pass@3 decreases. This corroborates our findings on Gaia2 where ERL mainly improved consistency through passˆ3 gains with modest pass@3 changes.

Telecom follows the opposite trend on every metric: success rate drops, passˆ3 falls to zero, yet pass@3 improves (0.625 vs. 0.575). We hypothesize that two design differences between Telecom and the other domains may explain this divergence. First, Telecom tasks are composed combinatorially from atomic subtasks, yielding a large configuration space that reduces the likelihood of any heuristic matching the specific subtask composition of a test task. Second, Telecom is a dual-control domain where the agent must guide a user who has their own tools, a randomly assigned persona, and variable device state. This makes the interaction less predictable than in single-control domains like Airline and Retail. As a result, a retrieved heuristic may improve the agent's diagnostic strategy on some trials, but fail on others when the user interaction unfolds differently.

Overall, results on τ^2-bench corroborate the Gaia2 findings in single-control domains, while the Telecom results highlight limitations around capturing user coordination strategies that could be an interesting direction for future work.

		Airline	Retail	Telecom	Overall
Success Rate	Baseline	36.7	43.3	**30.0**	36.7
	ERL	**38.3**	**47.5**	28.3	**38.0**
passˆ3	Baseline	20.0	12.5	**10.0**	**14.2**
	ERL	**25.0**	**17.5**	0.0	**14.2**
pass@3	Baseline	**55.0**	**80.0**	57.5	**64.2**
	ERL	50.0	75.0	**62.5**	62.5

Table 1: Per-domain results (in %) on τ^2-bench. Success rates are averaged over 3 trials per task. Evaluation is performed on the test split of each domain. ERL uses heuristics generated from the training split. **Bold**: best per group.

B GAIA2 EVALUATION SETUP

We conduct experiments on the Agents Research Environments (ARE) platform (Froger et al., 2025), which provides infrastructure for running and evaluating agents on the Gaia2 benchmark. ARE provides a default ReAct agentic scaffold which we use as our baseline. Each scenario defines a task along with a ground-truth trajectory, enabling automatic verification of task completion.

Test universes (22 and 25) were randomly selected from the available pool. As detailed in Table 2, we select GPT-5-mini to maintain an optimal balance between reasoning performance, inference latency, and computational cost.

ExpeL (Zhao et al., 2024) implementation. We sample a subset of training tasks matching the test set distribution (48 execution, 28 search). Following the original implementation, we collect success and failure experiences by running the Reflexion (Shinn et al., 2023) algorithm with GPT-5-mini for up to 3 retries. For insights extraction, we use a batch size of $L = 3$ successful trajectories. During inference, we retrieve 3 task-relevant few-shot examples using the Qwen3-Embedding-0.6B embedding model.

AutoGuide (Fu et al., 2024) implementation. To generate context-aware guidelines, we run Reflexion with GPT-5-mini for up to 3 retries on all training tasks, and run the guideline generation

Role	LLM
Agent	gpt-5-mini-2025-08-07
ARE LLM Judge	gpt-5-mini-2025-08-07
Heuristic generation	gpt-5-mini-2025-08-07
Heuristic retrieval	gpt-5.2-2025-12-11

Table 2: LLM model selection

process on scenarios with contrastive successful/failed trajectories. At test-time, the top 3 relevant guidelines are contextually retrieved and fed into the agent context after each *observation* ReAct turn. We use GPT-5-mini for guideline selection, extraction and context identification.

C Additional Experiments

C.1 Token-matched comparison with a few-shot baseline

We investigate whether heuristics outperform few-shot trajectories beyond token efficiency alone. Figure 5 demonstrates that for any number of experiences, using heuristics leads to higher success rates than raw trajectories. This holds when comparing the same number of experiences provided in-context as either heuristics or raw trajectories: heuristics yield a +5.5% gain in Execution at approximately 20 scenarios and a +23.8% gain in Search at 40 scenarios, where raw trajectories approach a regime where long-context performance starts to degrade. This confirms that distilling experiences into heuristics provides a superior learning signal compared to raw trajectory accumulation, while enabling substantially more experiences to fit within a fixed context budget.

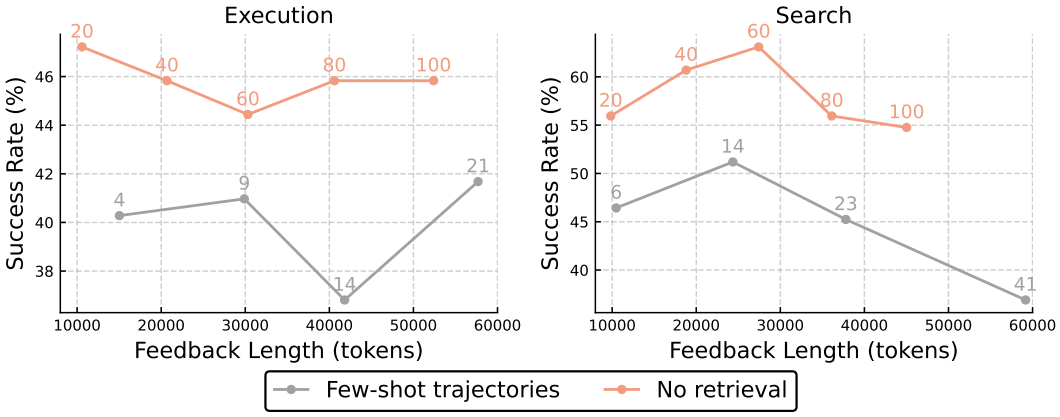


Figure 5: Success Rate over feedback token count for heuristic feedback and few-shot trajectories.

C.2 Heuristic retrieval

To determine the optimal retrieval approach and number of heuristics k to retrieve, we evaluate two methods: embedding-based and LLM-based retrieval. We limit our experiments to at most $k = 20$ heuristics, as this represents a practical upper bound: retrieving too many heuristics can be detrimental in complex scenarios, and the LLM-based retrieval approach becomes inefficient for $k > 20$. The experimental setup follows the general environment configuration described in Section B.

Embedding retrieval. We first evaluate embedding-based retrieval using the Qwen3-Embedding-0.6B model, computing cosine similarity between the test task and all training tasks, then selecting the top-k most similar heuristics. This approach effectively retrieves contextually similar tasks that

use the same tools (e.g., a calendar task retrieves all calendar-related training scenarios). However, it does not necessarily prioritize heuristics addressing similar error patterns. Table 3a shows the best overall performance is achieved with $k = 20$, yielding a 53.3% success rate.

LLM retrieval. To account for similarity between the test task and the content of the feedback itself, we evaluate LLM-based retrieval using GPT-5.2 as the ranker. The prompt in Figure 9 instructs the model to consider not only task similarity but also error patterns and the lessons articulated in each heuristic. Table 3b shows that LLM-based ranking outperforms embedding-based retrieval for most values of k, particularly on the Search split, with optimal performance again at $k = 20$ with a 56.1% success rate.

Importance of the reward signal. Reward signals may not always be available in real-world agentic tasks. We investigate a configuration where environment validation rewards are unavailable during heuristic generation, requiring the agent to infer the outcome of its trajectory and derive generalizable guidelines from this self-assessment. In this setting, the agent correctly identifies success or failure only 70% of the time, and the success rate of ERL drops to 51.2% (-4.8%). We note that this still exceeds the baseline (48.3%), indicating that ERL provides value even with imperfect reward signals, though accurate outcome feedback remains important for optimal performance.

	Success Rate (%)		
k	Execution	Search	Overall
1	43.1	**57.1**	50.1
2	49.3	44.1	46.7
3	43.8	**57.1**	50.5
4	50.0	<u>56.0</u>	<u>53.0</u>
5	**51.4**	47.6	49.5
10	45.8	<u>56.0</u>	50.9
15	47.2	51.2	49.2
20	<u>50.7</u>	<u>56.0</u>	**53.3**

(a) Embedding-based (Qwen3-Embedding-0.6B)

	Success Rate (%)		
k	Execution	Search	Overall
1	45.1	53.6	49.4
3	47.2	<u>58.3</u>	52.8
5	<u>49.3</u>	<u>58.3</u>	<u>53.8</u>
10	**51.4**	54.8	53.1
20	**51.4**	**60.7**	**56.1**

(b) LLM-based (GPT-5.2)

Table 3: Effect of retrieved heuristics count (k) by retrieval method. **Bold**: best; <u>underline</u>: second best.

D TOKEN COUNT AND COST ANALYSIS

		Tokens (M)			
Configuration	**Step**	**Input** (% cached)	**Output**	**Cost ($)**	**Avg. Turns**
Baseline	Scenario rollout	103.8 (82%)	4.3	15.27	16.6
ERL	Heuristic generation	7.5 (8%)	0.3	2.42	17.6
	Heuristic retrieval	1.0 (86%)	0.1	1.38	
	Scenario rollout	192.0 (88%)	3.9	17.60	
	Total	*200.5* (85%)	*4.3*	*21.40*	

Table 4: Breakdown of token usage and costs for the evaluation of the baseline and ERL agents. All experiments used the official OpenAI API with flex processing and models listed in Table 2.

We report total cumulative token counts and API costs for one full evaluation run on our Gaia2 test set, comparing the baseline ReAct agent against ERL with LLM retrieval of 20 heuristics per task. Input tokens for scenario rollout nearly double (+85%), primarily because retrieved heuristics ($\sim$20k tokens) are appended to the base system prompt ($\sim$12k tokens) at every turn, while the number of turns per scenario remains roughly constant (16.6 vs. 17.6). Thanks to prompt caching,

which absorbs a larger share of input tokens under ERL (88% vs. 82%), the cost increase for scenario rollout alone is a moderate +15%. Accounting for heuristic generation and retrieval, ERL incurs a 40% overall increase in API costs. Future work could explore more compact heuristic representations to preserve agent self-adaptation at lower computational overhead.

We do not measure latency, as it depends heavily on the model provider's API load. However, we note the additional LLM call involved in ERL to retrieve relevant heuristics is marginal for most Gaia2 tasks that span dozens of turns, but may be non-negligible for shorter tasks.

E ITERATIVE ERL

We also evaluate an iterative variant of ERL where the agent retrieves heuristics from the pool as it grows, rather than accumulating all heuristics first. Tasks are processed in batches; within each batch, the agent retrieves guidance from the current heuristic pool before execution, then adds newly generated heuristics to the pool (Algorithm 1). This creates a cumulative learning effect: early batches establish foundational knowledge from naive failures, while later batches benefit from this guidance and potentially encounter more nuanced errors. Comparing these two approaches tests whether learning from progressively improving trajectories offers advantages over learning from independent executions.

Algorithm 1 Iterative ERL

Input: N batches, size B, retrieval count k
Init: Pool $\mathcal{P} \leftarrow \varnothing$
for $i = 1$ **to** N **do**
  Sample batch $\mathcal{B}_i$
  for each $x \in \mathcal{B}_i$ **do**
    $C \leftarrow \textsc{Retrieve}(\mathcal{P}, x, k)$
    $\tau \leftarrow \textsc{Execute}(x, C)$
    $r \leftarrow \textsc{Reward}(\tau)$
    $h \leftarrow \textsc{Analyze}(x, \tau, r)$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(x, r, h)\}$
  end for
end for
Return: $\mathcal{P}$

	Source Tasks (%)		Test Tasks (%)		
Method	Execution	Search	Execution	Search	Overall
Baseline	—	—	43.1	53.6	48.3
ERL	42.0	53.8	**51.4**	**60.7**	**56.1**
Iterative ERL	**44.6**	**59.9**	46.5	54.8	50.7
Iterative ERL (only failures)	41.1	56.8	<u>47.9</u>	**60.7**	<u>54.3</u>

Table 5: Success rates on source and test tasks for ERL variants. Iterative ERL achieves higher source task performance but lower test generalization. **Bold**: best; <u>underline</u>: second best.

In Table 5, we report success rates on the test universes and on the source universes on which heuristics are accumulated. The scores reveal that iterative ERL achieves higher success rates than the standard version during accumulation but lower test performance (-5.4%). Several factors may explain this generalization gap. First, as iterative guidance improves performance, the agent may encounter fewer and narrower failure modes. The resulting heuristics could miss the breadth of naive mistakes that new tasks are likely to trigger. Second, heuristics from guided trajectories may be less transferable, as they reflect behavior shaped by prior guidance rather than independent discovery by the agent.

F HEURISTIC AND TRAJECTORY EXAMPLE

Figure 6 shows a sample heuristic. Heuristics follow a set structure with an analysis of why the agent succeeded or failed and a guideline with generalizable takeaways. We observe that guidelines often make reference to specific tools that the agent can call, and clarify idiosyncratic behaviors and errors that can steer future behavior.

Figure 7 shows an abridged trajectory where the agent explicitly refers to a heuristic during the completion of a new task. The heuristic, derived from a past failure where the agent forgot to delete an original event when rescheduling, prescribes a *safe reschedule* procedure: create the replacement event first, then delete the original. Although the new task involves replacing events rather than rescheduling, the agent recognizes the structural similarity and repeatedly applies the guideline (underlined), ensuring each original event is deleted after its replacement is created.

G HEURISTIC GENERATION AND RETRIEVAL PROMPTS

The heuristic generation prompt is provided in Figure 8. It takes as input a task, the trajectory of the agent, and the associated reward (success or failure). We guide the heuristics to emphasize constructive insights, enabling the agent to avoid repeating past mistakes and to build upon effective reasoning patterns when forming future reflections.

Heuristic retrieval at test time is performed by an LLM with structured outputs using the prompt in Figure 9. This version was selected after a human comparison with other variants containing more detailed rubrics and different scoring ranges. To handle retrieval a large pool of heuristics, future work could explore a two-stage process combining an embedding model to collect a large set of candidate heuristics, and the LLM to rerank them and select the top k.

Scenario ID: scenario_universe_27_9aqrhh

Task: Please order me an American Girl Pink Owl Bedroom Decor Girls Craft Kit that is of intermediate skill level. Also, can you reschedule my first upcoming dinner with colleagues to 10/20/2024 instead from 6 pm to 8 pm? Send an email to the attendees to inform them of the updated event. Additionally, remove all saved properties from my saved list that cost exactly $2000.

Reward: failure

1. **Analysis:**
   - **Breakpoint:** I attempted to call `Emails__send_email` using attendee names ("Sergei Kuznetsov", "Viktor Kuznetsov") instead of valid email addresses. The email tool requires real email addresses and threw an "Invalid email address" error. Although I recovered by looking up contacts and resending successfully, the initial incorrect tool call is the direct cause of the failure (invalid tool input).
2. **Learned Guideline:**
   - *Trigger:* When I need to send an email and my input is attendee names (from calendar events, meeting attendees, or other name-only lists).
   - *Action:* Resolve and validate recipient addresses before calling the email tool:
     - **Step 1:** For each attendee name, attempt `Contacts__get_contact` (exact) or `Contacts__search_contacts` (fuzzy) to retrieve an email address.
     - **Step 2:** If multiple contacts match, pick the best match deterministically (exact name match, same city, or most recent contact) and log the choice; if ambiguous or no match, ask the user for clarification or the correct email.
     - **Step 3:** Validate the resulting addresses have correct email format (contains '@' and a domain) before calling `Emails__send_email`.
     - **Step 4:** If `Emails__send_email` returns an "Invalid email address" error, parse the error to identify which recipient(s) failed, remove or correct only those entries, and retry rather than reissuing the original invalid request.

Rationale: Pre-resolving and validating recipients prevents invalid tool inputs, reduces retries, and keeps execution robust and auditable when sending messages on the user's behalf.

Figure 6: Heuristic example

TASK

I want to dedicate next month to young wine enthusiasts, so delete all events scheduled for next month that contain "Wine tasting" in the title and have one or more attendee over 45 years old. Schedule new events titled "Wine Awakening" to replace them with no attendees and the same timings. [. . .]

INJECTED HEURISTIC (EXCERPT)

Analysis: Root cause of failure: I created a new calendar event instead of actually "rescheduling" the existing event—I left the original event intact.

Learned Guideline:
Trigger: When instructed to "reschedule", "move", or "change the date/time" of an existing calendar event.
Action: I must explicitly locate the original `event_id`, then either (A) modify it using an edit API if available, or (B) if no edit API exists, create the new event **and delete the original event**. After doing either, verify the change by querying the calendar to confirm the original event is removed and the new event exists.

AGENT TRAJECTORY (EXCERPT)

[. . . the agent retrieves calendar events and contact records. . .]

Step 3. **Thought:** I can see a November 5 "Wine Tasting Event" that includes Vasile Munteanu (age 70), so it meets the criterion. Per the safe reschedule guideline, I should create the replacement event first with the same timings and no attendees, then delete the original.
Action: `Calendar__add_calendar_event` {title: "Wine Awakening", start: 2024-11-05 18:00, end: 2024-11-05 20:00, attendees: []}

Step 4. **Thought:** I created the replacement. Per the safe reschedule procedure, I should now delete the original event.
Action: `Calendar__delete_calendar_event` {event_id: ca87b8. . . }

[. . . the agent looks up the ages of attendees on the November 8 event. . .]

Step 7. **Thought:** The Nov 8 event includes August Blackwood (age 79), so it must be replaced. Following the safe-reschedule procedure, I'll create a new "Wine Awakening" event with the same details but no attendees.
Action: `Calendar__add_calendar_event` {title: "Wine Awakening", start: 2024-11-08 18:00, end: 2024-11-08 20:00, attendees: []}

Step 8. **Thought:** To complete the safe reschedule (create new event then remove the original), I will now delete the original event.
Action: `Calendar__delete_calendar_event` {event_id: eb831b. . . }

[. . . the same pattern repeats for Nov 12 and Nov 20. . .]

Figure 7: Example trajectory with explicit reference to heuristics

You are an AI agent engaging in self-reflection. You analyze your own trajectories to extract generalizable principles and guidelines that improve your future task execution across diverse scenarios. Your feedback is concise, strategic, and applicable beyond the specific task analyzed.

You are an intelligent agent operating in the ARE environment. You have just completed a task trajectory. Your goal is to perform a "Post-Mortem Analysis" to extract transferable lessons that will improve your reliability and reasoning on future tasks.

Context:
- Task Description: {task_info}
- Outcome: {validation_info}
- Trajectory: {trajectory_text}

Instructions: Analyze your performance based on the Outcome.

IF FAILURE:
1. **Pinpoint the Breakpoint:** Locate exactly where the logic failed or the tool was misused.
2. **Derive a Correction Rule:** Create a specific guideline that prevents this class of error.
   - *Bad:* "I should be more careful with files."
   - *Good:* "When a file path is unknown, I must use `ls -R` or `find` before attempting to `read_file` to avoid PathNotFound errors."

IF SUCCESS:
1. **Identify the "Winning Move":** What specific decision or tool usage made this efficient?
2. **Derive a Best Practice:** Create a rule to reinforce this behavior.
   - *Example:* "When asked to summarize data, aggregating it into a temporary file before processing is more robust than reading chunks."

Output Format: Construct your response to be appended to your future memory.
1. **Analysis:** (Briefly explain the cause of success or failure in this specific trace)
2. **Learned Guideline:** (A concise, imperative rule for future tasks. Focus on Tool Usage, Verification Loops, or Reasoning Steps.)
   - *Trigger:* [When/If I encounter Situation X...]
   - *Action:* [I must explicitly do Y...]

Reflect now:

Figure 8: Heuristic generation prompt

ROLE

You are an expert assistant specialized in selecting the most relevant heuristics to help an AI agent improve its performance on a given task. Heuristics are derived from past experiences (successful or failed) and contain a task description, an analysis of task completion, and learned guidelines.

GOAL

Your goal is to analyze and decompose the provided task into potential sub-tasks and action steps. Then, from the list of heuristics, identify the TOP {k} most relevant heuristics.

INPUT

You will be provided with:

1. A task description that the agent is going to perform.
2. A list of past experiences in the form of heuristics from previous tasks. Composed of:
   - Scenario ID
   - Task description
   - Reward (success or failure)
   - Heuristic text

OUTPUT

Your output should be a list of the TOP {k} scenario IDs of the most relevant heuristics that can help the agent improve its performance on the given task. You must also return a score between 0 and 100 and a brief justification for this score, based on the scoring criteria below.

HEURISTIC SELECTION

When selecting the most relevant heuristics, consider the following criteria:

- **Similarity of the task descriptions:** Look for heuristics where the task closely matches or relates to the current task.
- **Diversity of experiences:** Aim to include heuristics that cover a range of scenarios and challenges, providing a broad perspective.
- **Informative content:** Prioritize heuristics with a "Learned Guideline:" section that offer detailed insights, lessons learned, and actionable advice.

OUTPUT FORMAT

Your output should be a JSON object with the TOP {k} scenario IDs. For each ID, first provide a brief justification (1–2 sentences) of the score, then give a score (0–100):

```
{
"ID number 1":  ["rationale", score],
"ID number 2":  ["rationale", score],
...
"ID number k":  ["rationale", score]
}
```

LIST OF HEURISTICS:
{heuristics_list}

TASK TO COMPLETE:
{task}

Figure 9: Heuristic retrieval prompt