

From Private Memory to Collective Intelligence: Collaborative Test-Time Learning

Stanford CS224N Custom Project, Mentor: Mirac Suzgun

Jiaming Shen¹ Jiaxin Fang¹ Xinrui Jiang²

¹Department of Computer Science, Stanford University

²Department of Electrical Engineering, Stanford University
{shenjm, jiaxinfa, jiangxr}@stanford.edu

Abstract

Current multi-agent systems often suffer from "collective blindness," redundantly performing identical reasoning or tool-use errors. While the Dynamic Cheatsheet framework reduces repetitive errors for single agents, its self-evolving memory has not yet been generalized to collaborative ecosystems. We propose a "Hive Mind" architecture that transforms private memory into a shared, global asset. Our hierarchical system operates on two temporal scales: a Fast-Path Episodic Controller for real-time sharing of execution logs to eliminate redundant computation, and a Slow-Path Strategic Curator that distills successful trajectories into high-level strategic rules. In experiments on the GAIA benchmark using three parallel Magentic-One teams with Qwen3-8B, we counter-intuitively find that the memoryless multi-agent baseline achieves the highest reward, outperforming several memory-augmented variants including the FPEC-SPSC architecture. Our analysis suggests that extremely sparse rewards lead the controller to learn a conservative memory admission policy, revealing the critical vulnerability of shared memory to error cascading in complex multi-step reasoning tasks.

1 Key Information to include

Late Days: Jiaming Shen (4 remaining), Jiaxin Fang (4 remaining), and Xinrui Jiang (3 remaining); the team pools 3 late days and applies 3 late days to this report.

2 Introduction

Recent advances in large language models (LLMs) have shown that test-time scaling—allocating more compute for intermediate reasoning—can substantially improve performance on complex tasks [1]. Under this paradigm, agents produce explicit reasoning traces and tool-use trajectories before committing to a final solution. This has led to major gains on benchmarks where solutions require multi-step verification, such as web-based research and complex question answering [2, 3].

Despite these advances, current multi-agent systems (MAS) frequently suffer from a phenomenon we characterize as "collective blindness." In these environments, independent agents operating in parallel often redundantly perform the same reasoning errors or tool-use mistakes because they lack a mechanism to learn from the real-time experiences of their peers. While the Dynamic Cheatsheet (DC) [4] framework has introduced self-evolving memory to mitigate repetitive errors for individual agents, its sequential logic has not yet been successfully generalized to collaborative, multi-agent ecosystems. As a result, agents within a parallel workflow often waste computational resources on identical failed trajectories rather than building upon a shared foundation of collective intelligence.

In this paper, we propose a "Hive Mind" architecture that transforms private episodic memory into a shared, global asset. To bridge the gap between isolated sequential learning and real-time episodic

sharing, our system is structured as a hierarchical multi-agent framework operating on two distinct temporal scales, mimicking the cognitive functions of human "System 1" and "System 2" processes. By enabling both real-time memory exchange and asynchronous strategic distillation, our framework prevents redundant execution and allows agents to adapt their behavior dynamically during inference.

Our evaluation on the GAIA benchmark shows that the Multi-Agent (No Memory) baseline achieves the highest accuracy, outperforming several memory-augmented variants including our proposed FPEC–SPSC architecture. We find that the controller tends to learn a conservative storage policy due to extremely sparse rewards on difficult tasks, limiting the effectiveness of episodic memory sharing.

In sum, we make the following contributions in this paper:

- We propose the Hive Mind architecture, a hierarchical framework that enables real-time collaborative learning through shared episodic and strategic memory.
- We define the Fast-Path Episodic Controller (FPEC), a lightweight gatekeeper that identifies and caches high-utility reasoning steps to eliminate redundancy in parallel agent streams.
- We introduce the Slow-Path Strategic Curator (SPSC), which abstracts episodic hits into global strategic rules to improve the long-term reasoning accuracy of the agentic ecosystem.

3 Related Work

3.1 Agentic Systems and Multi-Agent Collaboration

LLM-based agents are autonomous systems designed to solve complex, open-ended problems by iteratively perceiving their environment, reasoning over current states, and invoking external tools [5, 6]. Early developments focused on augmenting single models with planning capabilities and API usage. Frameworks such as ReAct [7] and Toolformer [8] demonstrated that interleaving reasoning traces with external actions significantly extends an LLM’s capacity to complete long-horizon tasks. However, these single-agent systems operate strictly as sequential, isolated trajectories. Consequently, they are inherently fragile: a single hallucination or suboptimal tool invocation early on often derails the entire downstream execution, making them highly susceptible to error cascading.

To mitigate the brittleness of single-agent workflows, recent architectures have shifted toward multi-agent collaboration and parallel execution. Systems like Magentic-One [9] and M1-Parallel [10] instantiate multiple agent teams that independently explore diverse reasoning paths, subsequently aggregating their final outputs to yield more robust solutions. While this parallel generation improves overall reliability, it introduces a severe computational bottleneck: because these parallel teams execute in complete isolation, they frequently perform redundant operations. For instance, multiple agents might repeatedly query the same web pages or debug identical parsing errors, leading to massive inefficiencies in token and time consumption. This persistent "collective blindness" motivates our investigation into shared memory protocols that synchronize intermediate agent experiences.

3.2 Agent Memory System

Memory mechanisms have emerged as a core capability for foundation model-based agents, enabling persistence beyond single inference steps [11]. More recently, the ability of agents to leverage information derived from prior interactions, both within the current task and across previously completed tasks, has gained increasing significance and attention [12]. This capability has been formally distinguished from LLM memory, retrieval-augmented generation (RAG) [13], and context learning, and established as the distinct taxonomy of Agent Memory Systems.

Across diverse representational forms of agent memory, Token-level Memory has been most widely researched and applied, defined as storing information as externally accessible, discrete units. A growing body of work has therefore focused on how to structure token-level memory, determine what information should be stored, and design update and retrieval mechanisms that maximize memory utility for downstream reasoning. Specifically, DC-RS of DC variant organizes memory as flat text chunks, utilizing a curator and retrieval module to achieve test-time learning with adaptive memory. LTS [14] extends token-level memory to parallel agentic systems by introducing a global shared memory bank, where a lightweight RL-trained controller makes binary admission decisions on intermediate steps to enable cross-team reuse. However, both approaches employ flat memory archi-

tures failing to distinguish short-term episodic from longer-term semantic memory. Consequently, they are vulnerable to memory pollution and policy degeneration in low-reward regimes.

4 Approach

4.1 System Overview

Our goal is to enable multi-agent systems to continuously improve during inference by sharing useful intermediate knowledge across parallel tasks. Prior work has explored two complementary paradigms: sequential test-time learning, where a single agent accumulates experience across episodes [4], and real-time episodic sharing between parallel agents [14]. However, these approaches operate at different temporal scales and have not been unified in a single system.

To bridge this gap, our system is structured as a hierarchical multi-agent framework operating on two distinct temporal scales, mimicking cognitive "System 1" and "System 2" processes. As illustrated in Figure 1, our collaborative ecosystem shares raw execution logs to prevent redundant errors and asynchronously distills them into global strategic rules.

Our system consists of three major components:

- **Parallel Execution Layer:** multiple worker-agent teams execute tasks concurrently and generate raw reasoning trajectories.
- **Fast-Path Episodic Controller (FPEC):** a lightweight controller that performs real-time filtering of intermediate reasoning steps and stores high-utility knowledge into a shared episodic memory pool.
- **Slow-Path Strategic Curator (SPSC):** a higher-level reasoning module that periodically distills recurring successful patterns into global strategic rules.

Intuitively, FPEC acts as a fast reactive memory gate that allows agents to reuse useful intermediate results during ongoing execution, while SPSC performs slower reflective reasoning that extracts reusable high-level strategies from accumulated experience. This hierarchical design allows our system to support both immediate trajectory reuse and long-term strategic improvement.

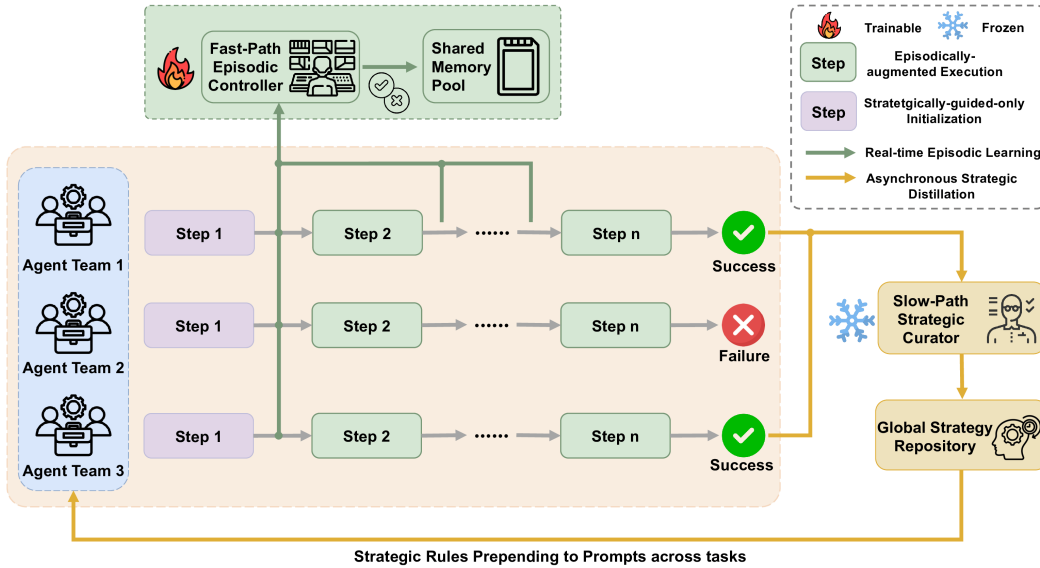


Figure 1: Overview of the proposed Hive Mind architecture. The Fast-Path Episodic Controller selectively caches intermediate execution logs into a global memory pool for real-time reuse, while the Slow-Path Strategic Curator asynchronously distills successful trajectories into strategic rules.

4.2 Parallel Execution Layer

At the lowest level, the system consists of K parallel worker-agent teams. Each team runs an LLM-based agent and is equipped with external tools including web search and code execution.

All intermediate outputs are streamed to the shared episodic memory subsystem. At each step, worker agents also query the shared memory pool for relevant cached results. If a matching entry is found, the agent can directly reuse the stored output without the need for re-computation; this enables the system to avoid redundant reasoning steps during parallel execution.

4.3 Fast-Path Episodic Controller (FPEC)

The Fast-Path Episodic Controller (FPEC) is responsible for determining which intermediate reasoning steps should be stored in the shared episodic memory pool.

Storing all intermediate steps indiscriminately would cause unbounded expansion of the memory repository and introduce noise during the retrieval process. To address this, the FPEC attempts to learn a selective storage strategy that prioritizes reasoning steps possessing high utility value.

Controller Architecture FPEC is implemented as a lightweight **Qwen3-0.6B** model, adapted from the memory controller architecture proposed in [14], and finetuned using LoRA adapters. Given the current inference step x_t , which consists of the step input, the step summary, and the step output (the step summary is generated by an agent that summarizes the step output), the controller predicts a storage probability:

$$p_\theta(z_t = 1 \mid x_t).$$

Here, $z_t \in \{0, 1\}$ indicates whether the current step should be stored in memory. If $z_t = 1$, the corresponding output is written to an *episodic memory pool*, $\mathcal{M} \leftarrow \mathcal{M} \cup \{x_t\}$. Worker agents can subsequently query this memory pool to retrieve and reuse previously stored outputs.

Controller Training The controller is trained using reinforcement learning to learn a storage policy that maximizes downstream task performance while maintaining a sparse and useful memory pool. In the following sections we describe two reward formulations: the original memory-controller objective used in prior work and our stabilized reward design that prevents policy collapse during training.

LTS-style Reward Design Following [14], the trajectory reward is computed using a partial-credit metric based on token overlap between predicted and reference answers, together with a bonus for the first team that completes the task: $R(\tau) = R_{\text{agg}}(\tau) + \lambda_{\text{first}} R_{\text{first}}(\tau)$.

To reduce policy gradient variance, we apply GRPO-style normalization and incorporate a usage-aware shaping signal rewarding steps whose stored outputs are later reused: $A_t^{(i)} = \frac{R(\tau^{(i)}) - \mu_R}{\sigma_R + \epsilon} + \beta \mathbb{I}(t \in U^{(i)} \wedge R(\tau^{(i)}) > 0)$.

The controller is optimized via $\mathcal{L}_{\text{policy}} = -\sum_t \log p_\theta(z_t \mid x_t) A_t$.

To discourage excessive storage, we add sparsity regularization $\mathcal{L}_{\text{sparse}} = \lambda_{\text{sparse}} \sum_t p_\theta(z_t = 1 \mid x_t)$, yielding $\mathcal{L} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{sparse}}$.

However, in practice we observed that this formulation can lead to unstable training dynamics. When the majority of trajectories receive very low rewards, the optimization process tends to minimize the sparsity penalty by collapsing the policy toward always predicting $z_t = 0$, resulting in an empty memory pool. Conversely, if the sparsity loss is omitted, the controller degenerates into always storing every step (i.e., $z_t = 1$), thereby effectively rendering the filtering mechanism inoperative.

Stabilized Reward Design To address the issue in LTS-style RL method, we introduce a stabilized reward formulation that provides richer supervision signals for intermediate reasoning behavior. Instead of relying solely on the final answer correctness, our reward function explicitly evaluates both the evidential quality of the trajectory and the usefulness of stored memory entries.

Episode-Level Reward For each trajectory τ , we define the total reward as $R(\tau) = R_{\text{agg}}(\tau) + \alpha_{\text{reward}} R_{\text{evidence}}(\tau) + \beta_{\text{reward}} R_{\text{memory}}(\tau) + \lambda_{\text{first}} R_{\text{first}}(\tau)$. Here, R_{agg} measures the correctness of the final aggregated answer using the partial-credit metric described earlier.

Evidence Reward Let $\mathcal{E}$ denote the set of important tokens and numerical values extracted from the reference answer and metadata annotations. We compute the fraction of these items that appear in the trajectory corpus (including the final answer and intermediate outputs): $R_{\text{evidence}}(\tau) = \frac{|\mathcal{E} \cap \text{Corpus}(\tau)|}{|\mathcal{E}|}$.

This reward encourages the agent to produce trajectories that explicitly surface relevant evidence rather than arriving at correct answers through spurious reasoning paths.

Memory Utility Reward Let U denote the set of stored steps whose outputs are retrieved during subsequent reasoning processes. Among these steps, we identify a subset of useful memories based on semantic overlap with the gold answer or reference reasoning steps. The memory reward is defined as $R_{\text{memory}}(\tau) = \frac{|U_{\text{useful}}|}{|U|}$. This reward explicitly encourages the controller to prioritize intermediate artifacts that provide reusable knowledge for other agents.

Admission-Rate Control In addition to improving the reward signal, we introduce an explicit admission-rate control mechanism to stabilize training.

Let $r = \frac{1}{T} \sum_{t=1}^T z_t$ denote the admission rate for a trajectory. We define a target interval $[r_{\text{low}}, r_{\text{high}}]$ representing a desirable memory utilization range. If the admission rate falls outside this interval, we apply an additional penalty: $\mathcal{L}_{\text{rate}} = \lambda_{\text{rate}} \left(\max(0, r - r_{\text{high}}) + \max(0, r_{\text{low}} - r) \right)$.

Entropy Regularization Finally, to encourage exploration, we include entropy regularization $\mathcal{L}_{\text{entropy}} = -\alpha H(p_\theta)$, where $H(p_\theta)$ denotes the entropy of the Bernoulli storage policy.

Final Training Objective The final controller objective combines the policy gradient loss with sparsity regularization and admission-rate control: $\mathcal{L} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{sparse}} + \mathcal{L}_{\text{rate}} + \mathcal{L}_{\text{entropy}}$.

Together, these modifications provide richer supervision for intermediate reasoning behavior and significantly improve the stability of controller training.

4.4 Slow-Path Strategic Curator (SPSC)

While FPEC captures useful low-level artifacts during execution, it does not explicitly learn reusable reasoning strategies. We therefore introduce the Slow-Path Strategic Curator (SPSC), which extracts high-level strategies from successful trajectories.

After each task, the curator evaluates the execution trajectory and generates a strategic summary when a useful reasoning pattern is identified. These summaries typically take the form of textual rules such as parsing instructions or task decomposition strategies. The strategies are stored in a global repository $\mathcal{S} = \{s_1, s_2, \dots\}$. During subsequent tasks, these strategies are prepended to worker-agent prompts, enabling agents to leverage prior reasoning patterns and avoid repeated reasoning failures.

4.5 Baselines

To evaluate the Hive Mind architecture and its hierarchical memory components, we compare our system against two baselines: (1) a **Multi-Agent (No Memory)** setup with independent teams, and (2) a **DC-RS** setup from [4] using original sequential memory update logic without a trained controller.

Implementation Details For the **DC-RS baseline**, we use the official implementation provided by the original Dynamic Cheatsheet paper [4]. The **Multi-Agent baseline** and the **Parallel Execution Layer** of our system are both built upon the Magentic-One framework [9]. Crucially, since the LTS paper does not provide an open-source code release, the **Fast-Path Episodic Controller (FPEC)** was engineered entirely from scratch. This intensive development process included reproducing the baseline LTS-style reinforcement learning environment, designing a novel stabilized reward formulation to handle sparse feedback, and seamlessly integrating these components into the complex Magentic-One asynchronous ecosystem to ensure robust parallel execution.

5 Experiments

5.1 Data

To evaluate our architecture in a setting that reflects realistic, general-purpose assistance, we utilize the GAIA benchmark [15]. GAIA targets tasks that are conceptually simple for humans but remain elusive for current AI systems. This performance gap highlights the benchmark’s focus on the robustness required for practical, real-world autonomy rather than specialized or professional knowledge. The benchmark comprises 466 human-curated, factoid questions categorized into three complexity levels:

- Level 1: Tasks requiring fewer than five steps and minimal tool usage, generally solvable by proficient LLMs without complex environmental feedback.
- Level 2: Problems demanding complex reasoning and the coordination of multiple tools, typically requiring five to ten steps.
- Level 3: The most challenging tasks, involving long-term planning and the advanced integration of various tools and evidence modalities.

While GAIA contains 466 questions in total, only the validation split (165 tasks) provides public ground-truth answers required for our RL reward. To maintain the focus of our evaluation on logical reasoning and multi-step tool coordination rather than local file system manipulation, we systematically excluded all tasks that explicitly referenced local file names or specific directory paths in the query, thereby avoiding edge cases related to local file handling that fall outside the scope of our collaborative architecture. The step distribution of the filtered dataset is provided in fig. 3. Following this filtration process, the remaining dataset was partitioned into a 70/30 split: 88 were allocated for the FPEC training phase, while the remaining 39 were reserved for evaluating the system’s performance on unseen tasks. A training example is shown in fig. 2.

GAIA Training Example
Query: If Eliud Kipchoge could maintain his record-making marathon pace indefinitely, how many thousand hours would it take him to run the distance between the Earth and the Moon its closest approach? Please use the minimum perigee value on the Wikipedia page for the Moon when carrying out your calculation. Round your result to the nearest 1000 hours and do not use any comma separators if necessary.
Answer: 17
Tools: A web browser, a search engine and a calculator.
Agent Steps:

- **Step 1:** Search for Eliud Kipchoge’s record pace (4min 37sec/mile).
- **Step 2:** Convert pace into decimal fractions of hours.
- **Step 3:** Retrieve the Moon’s minimum perigee in miles (225,623 miles) from Wikipedia.
- **Step 4:** Calculate total hours and round to the nearest 1000.

Figure 2: A representative training task from the GAIA benchmark.

5.2 Evaluation method

Performance is quantified using a Structured Partial Credit scoring system built upon the foundations of GAIA. This metric converts raw predictions and references into structured forms (dictionaries, lists, numbers, or strings) and applies type-specific rules:

- **Dictionaries & Lists:** Uses an F1-style score, recursively evaluating shared dictionary keys, and optimally aligning lists while penalizing length mismatches.
- **Numerical Data:** Full credit for exact matches, with partial credit decreasing linearly with base-10 log distance. Scores drop to zero if the prediction is off by an order of magnitude.

- General Text: Uses a token-level F1 score based on word overlap.

5.3 Experimental details

To implement our framework, we utilize Magentic-One [9] as our base agent architecture. During the training phase, we employ Gemini 2.5 Flash-lite to guarantee the generation of high-quality trajectories for the FPEC training stage, which is itself built upon the Qwen3-0.6B model. Our experimental environment consists of three parallel teams, with each team comprising a single Magentic-One agent. Each team generates three trajectories per task, following the GRPO training setup where multiple trajectories are sampled to compute normalized advantages.

The execution of these agents is governed by a strict step-limit policy to optimize both performance and resource consumption. Through empirical testing, we set a maximum of 10 steps for each agent, as our training samples (in fig. 3) indicate this depth is sufficient to resolve the majority of GAIA tasks. We specifically evaluated alternative constraints of 5 and 15 steps; however, a 5-step limit proved insufficient by prematurely cutting off reasoning before reaching a valid conclusion, while a 15-step limit resulted in excessive computational overhead without significant gains in accuracy. Due to hardware resource constraints, the FPEC was trained for one epoch, focusing the learning process on the high-utility transitions identified within the Gemini-generated traces. To prevent the controller from converging to trivial policies—either caching all transitions or suppressing memory storage entirely—we use the stabilized reward formulation (Section 4.3) with $\lambda_{\text{sparse}} = 0.1$, $\lambda_{\text{first}} = 0.1$, $\beta_{\text{usage}} = 0.15$, $\alpha_{\text{reward}} = 0.20$, and $\beta_{\text{reward}} = 0.20$, and train the controller with learning rate 3×10^{-5} .

5.4 Results

In our evaluation, we utilize Qwen3-8B as the base client for Magentic-One across all methods during inference, as our primary objective is to combine the FPEC + SPSC with worker agents to outperform standard architectures. Our evaluation maintains a consistent framework of three parallel teams, each generating individual trajectories that are subsequently used to produce a single aggregated answer.

Model Architecture	Inference Rewards
Multi-Agent (No Memory)	0.2153
DC-RS	0.2016
Multi-Agent (Full Memory)	0.1841
Multi-Agent (Full Memory + SPSC)	0.1371
FPEC-only	0.1364
FPEC-SPSC (Ours)	0.1353

Table 1: Inference accuracy on the GAIA test set ($n = 39$). Results are reported using our Structured Partial Credit metric across three parallel teams of Magentic-One agents.

Table 1 reveals that the Multi-Agent (No Memory) baseline achieves the highest inference rewards at 0.2153, followed by the Reward-Scaled Dynamic Cheatsheet (DC-RS) at 0.2016 and the Multi-Agent Full Memory at 0.1841. In contrast, our hierarchical FPEC-SPSC architecture and its standalone components recorded lower rewards, with FPEC-SPSC reaching 0.1353, while the FPEC-only and Full Memory + SPSC variants achieved 0.1364 and 0.1371, respectively. These results indicate that despite the integration of three parallel Magentic-One teams and the use of the trained FPEC to selectively cache high-utility assets for cross-team retrieval, the memory-augmented systems currently trail the memoryless baseline in solving complex GAIA tasks. We attribute this performance gap to the sparsity regularization during the training of the FPEC, where the controller learned a conservative policy in response to the near-zero rewards from worker agents on difficult traces. Additionally, SPSC generates only a small number of strategies in practice, many of which appear to be task-specific rather than broadly reusable.

6 Analysis

The experimental results indicate that while our hierarchical "Hive Mind" architecture provides a structured framework for collaborative learning, the current pipeline trails the memoryless multi-agent

baseline on the GAIA benchmark. This performance gap highlights several critical bottlenecks in the interaction between the FPEC and the SPSC.

The primary failure mode stems from the sparsity regularization applied during the FPEC training phase. In LTS’s design, a penalty term proportional to $p_\theta(z_t = 1|x_t)$ was applied at every step; this implied that the controller would incur a penalty whenever the predicted admission probability was high—even if the action actually sampled was "no-store." When task rewards were sparse, this penalty signal provided more stable optimization guidance compared to the high-variance, trajectory-level rewards. Consequently, the controller tends to minimize overall loss by reducing the admission probability, thereby leading to policy degradation (i.e., not storing any intermediate steps).

Although we introduced a stabilized reward design to mitigate this issue, the controller still exhibits a strong tendency to admit very few memory entries during inference. Because worker agents frequently receive near-zero rewards on high-difficulty GAIA tasks, optimization continues to favor minimizing the sparsity penalty over pursuing highly uncertain task rewards.

Additionally, to manage computational resource limits, we transitioned from Gemini 2.5 Flash-lite (used for generating high-quality training trajectories) to Qwen3-8B for the inference stage. Our objective was to evaluate whether the controller could transfer the "expert" strategies learned from the larger model to a system composed of smaller, more efficient agents. However, it appears that the Qwen3-8B agents lack the requisite capacity to consistently generate useful intermediate artifacts. Consequently, the FPEC may cache reasoning steps that are not only low-utility but potentially misleading to other teams, causing parallel agents to inject flawed reasoning into their own contexts and further degrading overall performance.

Training efficiency also limited system performance. Our controller was trained on only 88 samples for a single epoch, as each training update requires three parallel teams generating multiple trajectories of up to ten steps each. This results in a large number of LLM calls per step, making training extremely slow even on A10 GPUs and restricting the controller’s exposure to diverse reasoning patterns.

Finally, the effectiveness of the SPSC module is constrained by both trajectory quality and task diversity. Because many trajectories produced by Qwen3-8B contain incorrect reasoning, the curator often struggles to extract meaningful strategies. Moreover, GAIA tasks vary significantly in solution procedures, and many extracted strategies remain task-specific rather than broadly reusable, limiting the overall benefit of strategic distillation.

7 Conclusion

In this work, we introduced a hierarchical "Hive Mind" architecture unifying real-time episodic sharing with long-term strategic distillation in multi-agent systems. By structuring the framework into a Fast-Path Episodic Controller and a Slow-Path Strategic Curator, we transform isolated agent experiences into shared assets that mitigate the "collective blindness" inherent in parallel execution.

Our experimental evaluation on the GAIA benchmark revealed significant challenges in training lightweight controllers for complex, multi-step reasoning tasks. The FPEC demonstrated a persistent tendency toward policy degeneration. This behavior, coupled with the high computational cost of agentic training steps and the capacity limits of the Qwen3-8B worker agents, resulted in a system that currently trails memoryless baselines.

While the current pipeline was hindered by sparse rewards and training throughput constraints, the hierarchical design provides a scalable foundation for collective intelligence. Future research should focus on refining the admission-rate control mechanisms to stabilize the FPEC and leveraging more powerful hardware to enable multi-epoch training. By addressing these bottlenecks, we believe the Hive Mind approach can significantly enhance the efficiency and collaborative potential of autonomous agent ecosystems tackling complex reasoning tasks.

Team contributions

Jiaming Shen: Project conceptualization; Stabilized reward design and FPEC code implementation; RL training pipeline; SPSC implementation; Report writing

Jiaxin Fang: Project conceptualization; FPEC implementation; Multiagent baseline implementation; FPEC + SPSC inference implementation; Report writing

Xinrui Jiang: Project conceptualization; Data curation & analysis; DC variants adaption; hyperparameter optimization; FPEC-only inference; Report writing

References

- [1] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, et al. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.
- [2] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
- [3] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.
- [4] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.
- [5] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.
- [6] Xinzhe Li. A review of prominent paradigms for llm-based agents: Tool use (including rag), planning, and feedback learning. *arXiv preprint arXiv:2406.05804*, 2024.
- [7] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.
- [8] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.
- [9] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.
- [10] Enhao Zhang, Erkang Zhu, Gagan Bansal, Adam Fourney, Hussein Mozannar, and Jack Gerrits. Optimizing sequential multi-step tasks with parallel llm agents. *arXiv preprint arXiv:2507.08944*, 2025.
- [11] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.
- [12] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.
- [13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- [14] Joseph Fiorese, Parth Parag Kulkarni, Ashmal Vayani, Song Wang, and Mubarak Shah. Learning to share: Selective memory for efficient parallel agentic systems. *arXiv preprint arXiv:2602.05965*, 2026.
- [15] Grégoire Mialon, Clémentine Fourier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

A Appendix

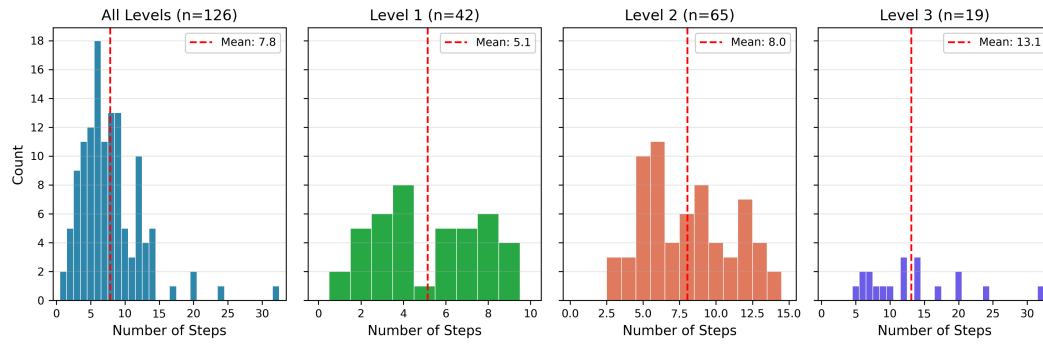


Figure 3: Distribution of annotated steps required for tasks in the filtered GAIA dataset.