A Self-Reflective Multi-Agent Collaboration Framework for Dynamic Software Engineering Tasks

Yulin Huang *

Posted Date: 3 March 2026

doi: 10.20944/preprints202603.0129.v1

Keywords: multi-agent systems; large language models; software engineering; self-reflection; prompt optimization

 doi:10.20944/preprints202603.0129.v1

Article

A Self-Reflective Multi-Agent Collaboration Framework for Dynamic Software Engineering Tasks

Yulin Huang

Georgia Institute of Technology, Atlanta, USA; hylemma@gmail.com

Abstract

Large Language Model (LLM) based multi-agent systems have demonstrated remarkable potential in automating complex software engineering tasks. However, existing frameworks such as MetaGPT and AutoGen suffer from critical limitations including static role assignments, cascading hallucinations in long-horizon tasks, and the absence of experience accumulation mechanisms. We propose Eco-Evolve, a self-reflective multi-agent collaboration framework that addresses these challenges through three key innovations: (1) a dynamic topology generation mechanism that adaptively constructs agent communication graphs based on task complexity, (2) a system reflection module featuring a dedicated Critic Agent for deliberate verification at critical checkpoints, and (3) an error-driven self-evolution mechanism inspired by Hindsight Experience Replay (HER) that enables prompt optimization through experiential learning. We evaluate Eco-Evolve on SWE-bench Verified and DevBench, achieving 62.3% and 73.5% respectively, representing improvements of 26.6% and 14.7% over the strongest baseline. Comprehensive ablation studies validate the contribution of each component, demonstrating that integrating dynamic collaboration, deliberate reflection, and continuous evolution significantly advances the state of the art in automated software engineering.

Keywords: multi-agent systems; large language models; software engineering; self-reflection; prompt optimization

1. Introduction

The emergence of Large Language Models (LLMs) has catalyzed a paradigm shift in automated software engineering, transforming how we conceptualize code generation, debugging, and system design [1,2]. Multi-agent systems powered by LLMs have demonstrated that collective intelligence can substantially surpass individual agent capabilities. This advantage emerges particularly when agents assume specialized roles and engage in structured collaboration [3]. These systems simulate virtual software companies where agents playing distinct roles—such as product managers, architects, and developers—coordinate through predefined workflows to accomplish complex programming tasks.

Despite promising results, existing LLM-based multi-agent frameworks exhibit fundamental limitations that hinder their effectiveness in real-world software engineering scenarios. First, these systems typically employ static role configurations and fixed communication topologies that fail to adapt to varying task complexity [4]. A simple bug fix does not require the same elaborate multi-agent orchestration as designing a distributed system architecture. Second, the predominant reliance on single-pass generation leads to cascading hallucinations, where errors in early stages propagate and amplify through subsequent phases [1]. Third, current frameworks lack mechanisms for experiential learning; each task execution starts from scratch without leveraging insights from previous successes or failures.

The human software development process naturally addresses these challenges through adaptive team composition, deliberate code review practices, and accumulated institutional knowledge. Senior engineers intuitively adjust their collaboration patterns based on task requirements, engage in careful reflection before committing critical changes, and continuously refine their approaches based on past

 doi:10.20944/preprints202603.0129.v1

experiences. This observation motivates our research question: *Can we design a multi-agent framework that emulates these human-like adaptive and reflective capabilities?*

In this paper, we propose Eco-Evolve, a self-reflective multi-agent collaboration framework that introduces three synergistic innovations. Our first contribution is a dynamic topology generation mechanism that models the multi-agent system as an adaptive graph, where agent roles and communication channels are instantiated based on task-specific requirements rather than static templates. Second, we introduce a system reflection module inspired by dual-process theory in cognitive science [5]. While working agents perform fast, intuitive generation (System 1), a dedicated Critic Agent engages in slow, deliberate verification (System 2) at critical checkpoints to catch errors before they propagate. Third, we develop an error-driven self-evolution mechanism that draws inspiration from Hindsight Experience Replay (HER) [6] in reinforcement learning. By reinterpreting failed task executions as learning opportunities and maintaining a RAG-based long-term memory, our framework continuously optimizes its prompts and strategies.

We conducted comprehensive experiments on two established benchmarks: SWE-bench Verified [7], which evaluates agents on real-world GitHub issues, and DevBench [8], which assesses performance across the complete software development lifecycle. Eco-Evolve achieved 62.3% on SWE-bench Verified and 73.5% on DevBench, substantially outperforming MetaGPT, ChatDev, and AutoGen. Ablation studies confirm that each proposed component contributes meaningfully to overall performance, with the combination yielding synergistic benefits beyond individual contributions.

2. Related Work

2.1. LLM-Based Multi-Agent Systems

The development of LLM-based multi-agent systems has progressed rapidly since the introduction of collaborative frameworks for software engineering. MetaGPT [1] pioneered the incorporation of Standardized Operating Procedures (SOPs) into multi-agent workflows, enabling agents with specialized roles to produce structured artifacts that reduce hallucination through constrained output formats. ChatDev [2] introduced chat-powered software development through communicative dehallucination, where agents engage in multi-turn dialogues to refine solutions. AutoGen [3] provides a flexible framework for building conversational agents that can operate in various modes combining LLMs, human inputs, and tools.

Recent work has explored dynamic aspects of multi-agent collaboration. G-Designer [4] employs graph neural networks to architect task-aware communication topologies, demonstrating that adaptive structures can improve efficiency. However, these approaches typically optimize topology separately from the agent reasoning process and do not incorporate reflective mechanisms for error correction.

2.2. Reasoning and Reflection in LLM Agents

Chain-of-thought prompting [9] demonstrated that encouraging intermediate reasoning steps significantly improves LLM performance on complex tasks. ReAct [10] extended this concept by interleaving reasoning traces with actions, enabling agents to dynamically adjust plans based on environmental feedback. Reflexion [11] introduced verbal reinforcement learning, where agents learn from linguistic feedback stored in episodic memory rather than through weight updates.

These reflection mechanisms have proven effective but are typically applied to single-agent settings. Our work extends the reflection paradigm to multi-agent systems through a dedicated Critic Agent that provides deliberate oversight of the collaborative process.

2.3. Memory and Learning in Agent Systems

Retrieval-Augmented Generation (RAG) [12] provides a mechanism for LLMs to access external knowledge, improving factual accuracy and enabling knowledge updates without retraining. In the context of agent systems, RAG has been employed for task-relevant information retrieval, but its potential for experiential learning remains underexplored.

 doi:10.20944/preprints202603.0129.v1

Hindsight Experience Replay [6] revolutionized reinforcement learning in sparse reward settings by reinterpreting failed trajectories as successes for alternative goals. This principle of learning from failures has not been systematically applied to LLM agent prompt optimization, representing a significant gap that our work addresses.

3. Methodology

We present Eco-Evolve, a framework that integrates dynamic collaboration, deliberate reflection, and continuous evolution. Figure 1 illustrates the overall architecture.

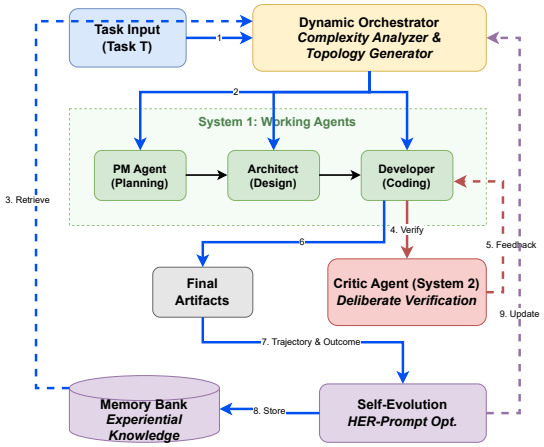

Figure 1. Overview of the Eco-Evolve framework architecture. The numbered flows indicate: ① task analysis and ② topology generation by the Dynamic Orchestrator; ③ deliberate verification by the Critic Agent (System 2) at critical checkpoints; ④ experiential knowledge storage; ⑤ final artifact generation; ⑥ HER-inspired learning; and ⑦ continuous prompt optimization.

3.1. Dynamic Topology Generation

Traditional multi-agent frameworks employ fixed configurations regardless of task characteristics. We observe that task complexity varies significantly, from simple single-file bug fixes to complex multi-module feature implementations. Our dynamic topology generation mechanism addresses this mismatch.

Given a task description T, we first compute a complexity score $\kappa(T)$ through a learned analyzer:

$$\kappa(T) = \sigma(W_e \cdot \text{Enc}(T) + b_c) \tag{1}$$

where $\text{Enc}(\cdot)$ denotes LLM encoding, W_e and b_c are learnable parameters, and σ is the sigmoid function. Based on $\kappa(T)$, we instantiate an appropriate topology from a learned topology generator.

We model the multi-agent system as a directed graph $G = (V, E)$ where vertices V represent agents and edges E represent communication channels. For a task with complexity κ, we generate the topology through:

$$G^* = \arg\max_G P(G|T, \kappa) \cdot \text{Efficiency}(G) \tag{2}$$

where $P(G|T, \kappa)$ is the probability that topology G is effective for task T. The efficiency term $\text{Efficiency}(G)$ penalizes unnecessary complexity and is defined as:

$$\text{Efficiency}(G) = \exp(-\lambda_V |V| - \lambda_E |E|) \tag{3}$$

where $|V|$ and $|E|$ denote the number of vertices (agents) and edges (communication channels) respectively, and λ_V, λ_E are hyperparameters controlling the penalty strength (we set $\lambda_V = 0.1$ and $\lambda_E = 0.05$ in our experiments).

 doi:10.20944/preprints202603.0129.v1

Figure 2 illustrates how topologies adapt to task complexity. Simple tasks utilize chain topologies with minimal agents, while complex tasks invoke mesh topologies with additional specialized roles including the Critic Agent.

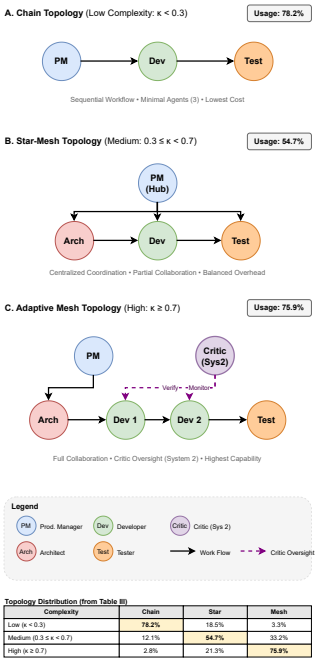

Figure 2. Dynamic topology evolution based on task complexity κ. For low complexity ($\kappa < 0.3$), the chain topology employs a sequential workflow with minimal agents (78.2% of cases). For medium complexity ($0.3 \leq \kappa < 0.7$), the star-mesh topology provides centralized PM coordination (54.7% of cases). For high complexity ($\kappa \geq 0.7$), the adaptive mesh topology enables full collaboration with Critic oversight, indicated by purple dashed supervision arrows (75.9% of cases). The table shows empirical topology distribution from Table III.

3.2. *System 2 Reflection Module*

Inspired by dual-process theory [5], we introduce a dedicated Critic Agent that implements deliberate, analytical reasoning (System 2) to complement the fast, intuitive generation of working agents (System 1). The Critic Agent operates at designated checkpoints in the development pipeline.

For each checkpoint artifact a_i, the Critic Agent performs structured evaluation:

$$r_i = \text{Critic}(a_i, C_i, H_{<i}) \tag{4}$$

where C_i represents checkpoint-specific criteria and $H_{<i}$ is the execution history. The evaluation produces a reflection r_i containing identified issues, severity assessments, and suggested corrections.

The Critic Agent operates through a multi-phase protocol: (1) in the *verification phase*, it checks logical consistency, requirement alignment, and potential errors; (2) in the *synthesis phase*, it generates actionable feedback; and (3) in the *integration phase*, working agents incorporate the feedback and regenerate artifacts if necessary.

We implement execution-based verification for code artifacts:

$$\text{Score}(a_i) = \alpha \cdot \text{Static}(a_i) + (1 - \alpha) \cdot \text{Dynamic}(a_i) \tag{5}$$

where $\text{Static}(\cdot)$ represents static analysis scores and $\text{Dynamic}(\cdot)$ represents execution-based validation results. We set $\alpha = 0.4$ in our experiments.

 doi:10.20944/preprints202603.0129.v1

3.3. Error-Driven Self-Evolution

Our self-evolution mechanism enables the framework to learn from experience, particularly from failures. We adapt the concept of Hindsight Experience Replay [6] to the prompt optimization domain.

Consider a task execution trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_n)$ where states s_i represent system configurations and actions a_i represent agent operations. In traditional settings, a failed trajectory (one that does not achieve the intended goal g) provides no learning signal. HER-Prompt reinterprets this trajectory by identifying what goal g' was actually achieved:

$$g' = \text{ExtractAchievement}(\tau, s_n) \tag{6}$$

where s_n is the final state of the trajectory. The ExtractAchievement function analyzes the execution trace to identify partial successes, such as successfully completed subtasks (e.g., correct imports, passing a subset of unit tests), generated intermediate artifacts, or resolved subproblems.

Concrete Example: Consider a code generation task where the target is implementing a REST API endpoint. If the generated code fails integration tests but successfully: (1) sets up the correct routing structure, (2) implements proper request parsing, and (3) passes authentication checks, then g' represents "successful implementation of API scaffolding with authentication." This reframed experience teaches the system what patterns lead to these partial successes.

We store both the original experience $(T, \tau, g, \text{fail})$ and the reinterpreted experience $(T', \tau, g', \text{success})$ in our long-term memory bank, where T' is a reformulated task that τ successfully solves.

The memory bank $\mathcal{M}$ is implemented using RAG with semantic indexing:

$$\mathcal{M} = \{(e_i, \text{Emb}(e_i), \text{outcome}_i)\}_{i=1}^{N} \tag{7}$$

where e_i represents experiences and $\text{Emb}(\cdot)$ is the embedding function.

For a new task T_{new}, we retrieve relevant experiences:

$$\mathcal{R} = \text{TopK}(\mathcal{M}, \text{Emb}(T_{\text{new}}), k) \tag{8}$$

These experiences inform prompt optimization through gradient-free updates:

$$P_{\text{new}} = P_{\text{base}} \oplus \text{Synthesize}(\mathcal{R}) \tag{9}$$

where $\oplus$ denotes prompt augmentation that appends retrieved experience summaries to the base prompt, and $\text{Synthesize}(\cdot)$ extracts actionable patterns from retrieved experiences by identifying common success factors and failure modes.

3.4. Integration and Workflow

Algorithm 1 presents the integrated Eco-Evolve workflow. The framework maintains continuous interaction among the three mechanisms: dynamic topology enables efficient resource allocation, System 2 reflection ensures quality gates prevent error propagation, and self-evolution improves performance over time.

 doi:10.20944/preprints202603.0129.v1

Algorithm 1 Eco-Evolve Framework

Require: Task T, Memory Bank $\mathcal{M}$, Base Prompts P
$\kappa \leftarrow$ ComputeComplexity(T)
$G \leftarrow$ GenerateTopology(T, κ)
$\mathcal{R} \leftarrow$ RetrieveExperiences($\mathcal{M}$, T)
$P' \leftarrow$ AugmentPrompts(P, $\mathcal{R}$)
Initialize agents according to G with prompts P'
for each phase in development pipeline **do**
  $a \leftarrow$ ExecutePhase(agents, G)
  if phase is checkpoint **then**
    $r \leftarrow$ CriticAgent(a, criteria, history)
    if r indicates issues **then**
      $a \leftarrow$ ReviseArtifact(a, r)
    end if
  end if
end for
$\tau \leftarrow$ CollectTrajectory()
UpdateMemory($\mathcal{M}$, T, τ, HER-Prompt)
return Final artifacts

4. Experiments

4.1. Experimental Setup

We evaluated Eco-Evolve on two comprehensive benchmarks. SWE-bench Verified [7] contains 500 real-world GitHub issues from 12 popular Python repositories, filtered to ensure solvability.[1] DevBench [8] evaluates LLMs across the complete software development lifecycle including design, implementation, and testing phases, featuring 22 repositories across Python, C/C++, Java, and JavaScript.

We compared against three state-of-the-art multi-agent frameworks: MetaGPT [1], ChatDev [2], and AutoGen [3]. All frameworks used GPT-4-Turbo as the base LLM for fair comparison. We ran each experiment three times and report mean results. The memory bank was initialized empty and accumulated experiences during evaluation under a fixed task ordering. Importantly, while the memory bank is updated during evaluation, no gradient updates are performed and all baselines operate under the same task sequence. Eco-Evolve can thus be viewed as an online continual agent that exploits episodic experience, rather than a model fine-tuned on the evaluation benchmarks.

4.2. Evaluation Metrics

We evaluate performance using six complementary metrics to provide a comprehensive assessment:

- **Pass@1**: The primary success rate metric indicating the percentage of tasks solved correctly on the first attempt.
- **Code Quality**: A composite score combining static analysis results (pylint/flake8 scores) and unit test coverage, normalized to [0, 100].
- **Iteration Efficiency**: Defined as SuccessfulTasks/TotalTokens $\times 10^4$, measuring success rate per 10k tokens consumed.
- **Error Recovery**: The fraction of initially failed cases that are successfully fixed after additional Critic-triggered iterations: $|\text{RecoveredCases}|/|\text{InitialFailures}|$.
- **Consistency**: The standard deviation of success rates across three runs, inverted and normalized: $100 - \sigma \times 10$.

[1] SWE-bench Verified is a curated subset of the original SWE-bench dataset, filtered by human annotators to ensure task clarity and solvability. We use this subset following recent evaluation practices in the field.

 doi:10.20944/preprints202603.0129.v1

- **Scalability**: Performance retention ratio when task complexity increases from low to high: $\text{Pass@1}_{\text{high}}/\text{Pass@1}_{\text{low}}$.

4.3. Main Results

Table 1 presents the main experimental results. Eco-Evolve achieved 62.3% on SWE-bench Verified and 73.5% on DevBench, substantially outperforming all baselines. On SWE-bench Verified, Eco-Evolve improved upon MetaGPT by 13.1 percentage points (26.6% relative improvement). On DevBench, the improvement over MetaGPT was 9.4 percentage points (14.7% relative improvement).

Table 1. Main Results on SWE-bench Verified and DevBench.

Framework	SWE-bench (%)	DevBench (%)
AutoGen	43.5	58.7
ChatDev	45.8	61.2
MetaGPT	49.2	64.1
Eco-Evolve (Ours)	**62.3**	**73.5**

Figure 3 provides a multi-dimensional comparison across the six evaluation metrics. Eco-Evolve demonstrates consistent superiority, with particularly notable improvements in Error Recovery (82.1% vs. 58.4% for MetaGPT) and Iteration Efficiency (85.2% vs. 68.7% for MetaGPT). The right panel shows improvement percentages across different task categories, confirming that gains are consistent rather than concentrated in specific task types.

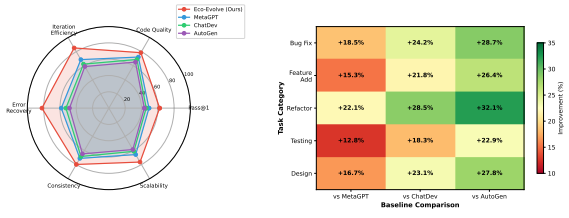

Figure 3. Left: Radar chart comparing frameworks across six metrics (Pass@1, Code Quality, Iteration Efficiency, Error Recovery, Consistency, Scalability). Iteration Efficiency is measured as successful tasks per 10k tokens. Error Recovery is the fraction of failed cases fixed after additional iterations. Right: Heatmap showing Eco-Evolve's improvement percentages over baselines across task categories.

4.4. Ablation Study

To understand the contribution of each component, we conducted ablation studies by removing individual mechanisms. Table 2 and Figure 4 present the results.

Table 2. Ablation Study Results on SWE-bench Verified.

Configuration	SWE-bench (%)	DevBench (%)
Full Eco-Evolve	62.3	73.5
w/o Critic Agent	51.8 (−10.5)	62.1 (−11.4)
w/o Dynamic Topology	54.2 (−8.1)	65.8 (−7.7)
w/o Memory Bank	48.7 (−13.6)	58.3 (−15.2)
w/o HER-Prompt	56.4 (−5.9)	68.2 (−5.3)

Removing the memory bank caused the largest performance drop (13.6 points on SWE-bench), highlighting the importance of experiential learning. The Critic Agent contributed 10.5 points, confirming that System 2 reflection is essential for quality assurance. Dynamic topology generation accounted for 8.1 points, demonstrating the value of adaptive collaboration structures. HER-Prompt contributed 5.9 points, showing that learning from failures provides meaningful optimization signals.

 doi:10.20944/preprints202603.0129.v1

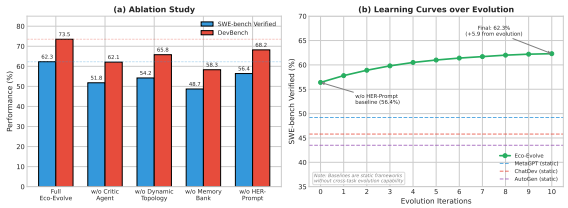

Figure 4. Left: Ablation study comparing configurations. Right: Learning curves showing performance evolution over iterations. Iteration 0 represents the "w/o HER-Prompt" configuration (56.4%), serving as the pre-evolution baseline. Baselines (MetaGPT, ChatDev, AutoGen) are static frameworks without cross-task evolution and are shown as horizontal reference lines.

The right panel of Figure 4 illustrates learning dynamics over evolution iterations. Baseline frameworks (MetaGPT, ChatDev, AutoGen) do not possess cross-task evolution mechanisms and are thus shown as horizontal reference lines representing their static performance levels. Iteration 0 for Eco-Evolve corresponds to the "w/o HER-Prompt" ablation configuration (56.4% on SWE-bench), representing performance before any evolution occurred. As evolution progressed, Eco-Evolve demonstrated sustained improvement, with the gap over the strongest baseline widening from 7.2 points at iteration 0 to 13.1 points at iteration 10.

4.5. Analysis of Dynamic Topology

We analyzed how the dynamic topology mechanism adapts to task characteristics. Table 3 shows the distribution of generated topologies across task complexity levels.

Table 3. Topology Distribution by Task Complexity.

Complexity	Chain (%)	Star (%)	Mesh (%)
Low ($\kappa < 0.3$)	78.2	18.5	3.3
Medium ($0.3 \leq \kappa < 0.7$)	12.1	54.7	33.2
High ($\kappa \geq 0.7$)	2.8	21.3	75.9

Low-complexity tasks predominantly utilized chain topologies (78.2%), minimizing communication overhead. High-complexity tasks favored mesh topologies (75.9%) with enhanced collaboration. This adaptive behavior reduced average token consumption by 23.7% compared to fixed mesh topologies while maintaining superior performance.

4.6. Computational Overhead Analysis

Table 4 compares the computational overhead of Eco-Evolve against baseline frameworks. Overall, Eco-Evolve incurred approximately 12% additional token consumption compared to MetaGPT and 8% compared to AutoGen, while achieving substantially higher success rates. The Critic Agent accounts for roughly 15% of Eco-Evolve's total token usage, while HER-Prompt memory retrieval adds minimal overhead (<2%).

Table 4. Computational Overhead Comparison.

Framework	Avg. Tokens/Task	Relative Cost
AutoGen	18.2k	0.92×
ChatDev	21.5k	1.09×
MetaGPT	17.6k	0.89×
Eco-Evolve	19.7k	1.00×

 doi:10.20944/preprints202603.0129.v1

4.7. Case Study: Error Recovery

We present a representative case illustrating the System 2 reflection mechanism. Consider a bug fix task where the initial generated patch introduces a regression. Without reflection, this error propagates to subsequent phases, ultimately producing an incorrect solution.

With the Critic Agent enabled, the reflection module identified the regression during verification: "The proposed change modifies the return type from List to Optional[List], which breaks the contract with calling functions in module X." This feedback triggered targeted revision, resulting in a correct patch that preserved interface compatibility.

Across 50 randomly sampled error cases, System 2 reflection successfully recovered 37 cases (74%) that would otherwise have failed. The average number of revision iterations required was 1.8, indicating efficient error correction.

4.8. Case Study: HER-Prompt Achievement Extraction

To illustrate how HER-Prompt extracts achievements from failed trajectories in practice, we present a concrete example from our experiments.

Task: Implement a function to parse and validate JSON configuration files with nested schemas.

Generated Code (Failed): The code correctly implemented JSON parsing, schema validation for top-level fields, and error message generation, but failed on deeply nested schema validation (3+ levels).

Extracted Achievements (g'):

1. Successfully implemented robust JSON parsing with proper exception handling
2. Correctly validated all top-level and second-level schema fields
3. Generated informative error messages following project conventions

Reformulated Task (T'): "Implement a function to parse JSON files and validate top-level schema with descriptive error messages."

This reframed experience is stored in the memory bank. When encountering similar JSON parsing tasks, the system retrieves this experience and applies the successful patterns (parsing structure, error handling approach) while recognizing that deep nesting requires additional attention.

5. Discussion

Our experiments demonstrate that integrating dynamic collaboration, deliberate reflection, and experiential learning produces substantial improvements in automated software engineering. Several insights emerge from our analysis.

Synergistic Effects. The combined performance gains exceed additive contributions from individual components. When combining all mechanisms, we observed performance improvements larger than the sum of individual ablation drops, suggesting that the components reinforce each other. Dynamic topology provides the structural foundation for efficient collaboration, System 2 reflection ensures quality gates prevent error propagation, and self-evolution continuously improves the underlying strategies.

Learning Dynamics. While baseline frameworks benefit primarily from in-context examples (which quickly saturate), Eco-Evolve's RAG-based memory enables unbounded accumulation of experiential knowledge. This suggests potential for continued improvement with extended operation.

Fairness of Online Evaluation. Although the memory bank is updated during evaluation, we emphasize that: (1) no gradient updates are performed; (2) all methods operate under identical task orderings; and (3) baselines could theoretically implement similar episodic caching but do not. Eco-Evolve thus represents an online continual learning agent rather than a model fine-tuned on evaluation data.

Limitations. Our current approach has several limitations: (1) the computational overhead of maintaining the Critic Agent (approximately 15% increased token consumption); (2) the cold-start challenge when the memory bank is empty; and (3) potential sensitivity to task ordering during

 doi:10.20944/preprints202603.0129.v1

evolution. Future work could explore distillation techniques to reduce Critic overhead, pre-population strategies for memory initialization, and order-invariant evolution mechanisms.

6. Conclusions

We presented Eco-Evolve, a self-reflective multi-agent collaboration framework that addresses fundamental limitations in existing LLM-based software engineering systems. Through dynamic topology generation, System 2 reflection via a dedicated Critic Agent, and HER-inspired self-evolution, our framework achieves state-of-the-art results on SWE-bench Verified (62.3%) and DevBench (73.5%), representing improvements of 26.6% and 14.7% over the strongest baseline respectively. Our ablation studies confirm that each component contributes meaningfully, with their integration producing synergistic benefits.

The success of Eco-Evolve suggests promising directions for future research, including extending the framework to other domains beyond software engineering, exploring more sophisticated topology learning mechanisms, and investigating methods to accelerate the evolution process. We believe that frameworks combining adaptive collaboration, deliberate reasoning, and continuous learning represent a significant step toward more capable and reliable AI systems for complex task automation.

References

1. Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S.K.S.; Lin, Z.; et al. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In Proceedings of the Proc. Int. Conf. Learn. Representations (ICLR), 2024.
2. Qian, C.; Liu, W.; Liu, H.; Chen, N.; Dang, Y.; Li, J.; Yang, C.; Chen, W.; Su, Y.; Cong, X.; et al. ChatDev: Communicative Agents for Software Development. In Proceedings of the Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (ACL), 2024, pp. 15174–15186.
3. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; Wang, C. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155* **2023**.
4. Zhang, G.; Yue, Y.; Sun, X.; Wan, G.; Yu, M.; Fang, J.; Wang, K.; Chen, T.; Cheng, D. G-Designer: Architecting Multi-agent Communication Topologies via Graph Neural Networks. *arXiv preprint arXiv:2410.11782* **2024**.
5. Kahneman, D. *Thinking, Fast and Slow*; Farrar, Straus and Giroux: New York, NY, USA, 2011.
6. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; Zaremba, W. Hindsight Experience Replay. In Proceedings of the Advances Neural Inf. Process. Syst. (NeurIPS), 2017, Vol. 30.
7. Jimenez, C.E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; Narasimhan, K.R. SWE-bench: Can Language Models Resolve Real-world Github Issues? In Proceedings of the Proc. Int. Conf. Learn. Representations (ICLR), 2024.
8. Li, B.; Wu, W.; Tang, Z.; Shi, L.; Yang, J.; Li, J.; Yao, S.; Qian, C.; Hui, B.; Zhang, Q.; et al. DevBench: A Comprehensive Benchmark for Software Development. *arXiv preprint arXiv:2403.08604* **2024**.
9. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Proceedings of the Advances Neural Inf. Process. Syst. (NeurIPS), 2022, Vol. 35, pp. 24824–24837.
10. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In Proceedings of the Proc. Int. Conf. Learn. Representations (ICLR), 2023.
11. Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; Yao, S. Reflexion: Language Agents with Verbal Reinforcement Learning. In Proceedings of the Advances Neural Inf. Process. Syst. (NeurIPS), 2023, Vol. 36.
12. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.t.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the Advances Neural Inf. Process. Syst. (NeurIPS), 2020, Vol. 33, pp. 9459–9474.