AgenticNet: Rethinking Multi-agent System Architectures with LLM-based Networks

Anonymous authors
Paper under double-blind review

Abstract

Multi-agent systems (MAS) enhance the capabilities of single LLM agents by leveraging collaboration and specialization. However, existing designs often rely on ad-hoc coordination strategies, lacking a principled architecture that integrates reasoning, communication, and adaptation. This limitation makes it difficult to scale multi-agent systems in a way that is both effective and interpretable. To address this challenge, we take inspiration from the architecture of neural networks to rethink MAS design. We treat each LLM-based agent as a computational unit and organize them into layered structures, analogous to neurons and layers, which we call AgenticNet. In this framework, lower layers act as planners that decompose problems, intermediate layers function as executors that advance reasoning step by step, and upper layers serve as synthesizers that verify consistency and deliver final decisions. Information propagates forward through the layers, while adaptation is guided by layer-level prompt updaters and a global prompt supervisor that refine agent behavior based on task loss, serving as an analogue to backpropagation. We conduct extensive experiments on five benchmarks, including AIME24, GSM8K, MATH500, HumanEval, and MBPP. Across all tasks, AgenticNet consistently outperforms both single-agent baselines and existing multi-agent systems, demonstrating its effectiveness as a scalable architecture for multi-agent.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in reasoning (Wei et al., 2022), problem solving (Wei et al., 2022), and tool use (Schick et al., 2023), but single-agent deployments often struggle with scalability, robustness, and specialization. To address these limitations, recent research has explored multi-agent systems (MAS) (Park et al., 2023), where multiple LLM agents collaborate through interaction and role differentiation. Such systems can enhance performance by dividing labor, integrating diverse skills, and enabling emergent behaviors beyond those achievable by a single agent.

Existing MAS architectures can be broadly categorized into four types. Equi-level structures (Du et al., 2023) place all agents at the same hierarchy, encouraging collective decision-making but lacking mechanisms to resolve conflicts or enforce coherent strategies. Hierarchical structures (Hong et al., 2024a) rely on leader–follower dynamics, which simplify coordination but introduce single points of failure and reduce system robustness. Nested or hybrid structures (Li et al., 2023b) combine equi-level and hierarchical subsystems, providing flexibility but greatly increasing design complexity and making agent interactions harder to control. Dynamic structures (Liu et al., 2024) allow agents to adapt roles based on context, which improves flexibility but makes behaviors unstable and difficult to interpret. The typical MAS architectures are shown in Fig. 1.

Despite their differences, these paradigms share several fundamental limitations. First, planning remains largely ad hoc, as current systems lack principled mechanisms to decompose complex tasks and coordinate contributions across heterogeneous agents. Second, memory and adaptation are weak: MAS struggle to maintain long-term consistency across extended reasoning horizons and lack systematic methods to refine agent behavior when errors accumulate. Third, scalability and interpretability remain major obstacles,

since interactions among agents become increasingly difficult to control as the number of agents grows, and collective behavior often appears opaque and hard to audit. Consequently, while these categories demonstrate the diversity of MAS design, existing approaches remain fragmented and lack a principled architectural foundation.

To overcome these limitations, we propose AgenticNet, a network inspired by neural network architectures that provides a principled design for multi-agent systems. In AgenticNet, each LLM-based agent is treated as a computational unit, analogous to a neuron, and agents are organized into layered structures to enable compositional reasoning. For the planning challenge, lower layers act as planners that decompose tasks, intermediate layers function as executors that advance reasoning step by step, and upper layers serve as synthesizers that verify consistency and produce final outputs. To address memory and adaptation, we introduce layer-level prompt updaters and a global prompt supervisor that iteratively refine agent behavior based on task loss, serving as an analogue to backpropagation. To improve scalability and interpretability, the layered design of AgenticNet provides modularity and transparency, allowing reasoning chains to be structured, auditable, and easier to analyze. The AgenticNet shown in Fig. 2.

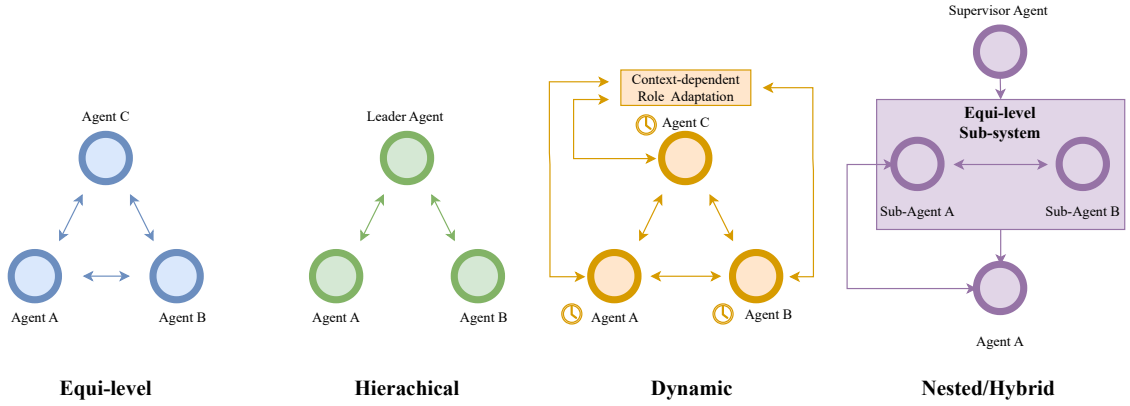

Figure 1: The architectures of typical MAS architectures

2 Related Work

2.1 Multi-Agent System Architectures.

LLM-based agent research has evolved from single-agent designs to increasingly sophisticated multi-agent systems (MAS). Early approaches like Chain-of-Thought and ReAct Wei et al. (2022); Yao et al. (2023) used a single LLM for reasoning and acting, but were constrained by the capacity of one model to handle all steps. To overcome these constraints, MAS introduce multiple agents that communicate and collaborate, which has been shown to improve task success Wang et al. (2025). Within MAS, two major directions have emerged: general-purpose architectures, which aim to address a wide range of tasks across domains and coordinate agents either statically or dynamically Wu et al. (2024); Li et al. (2023a); Hong et al. (2024b); Chen et al. (2023b), and application-specific systems, which design agent roles and workflows explicitly for domains such as code generation Qian et al. (2023); Huang et al. (2023), bio-inspired discovery Ghafarollahi & Buehler (2025; 2024), or e-commerce Li et al. (2023c); Zhao et al. (2023). However, most existing architectures remain static in role assignment and workflow design, motivating research into more adaptive, self-evolving MAS.

2.2 Self-evolving multi-agent systems

Self-evolving MAS aim to overcome this limitation by dynamically adjusting prompts, workflows, or even agent topologies during execution Hu et al. (2024a); Zhang et al. (2025b); Yuan et al. (2024).For instance,

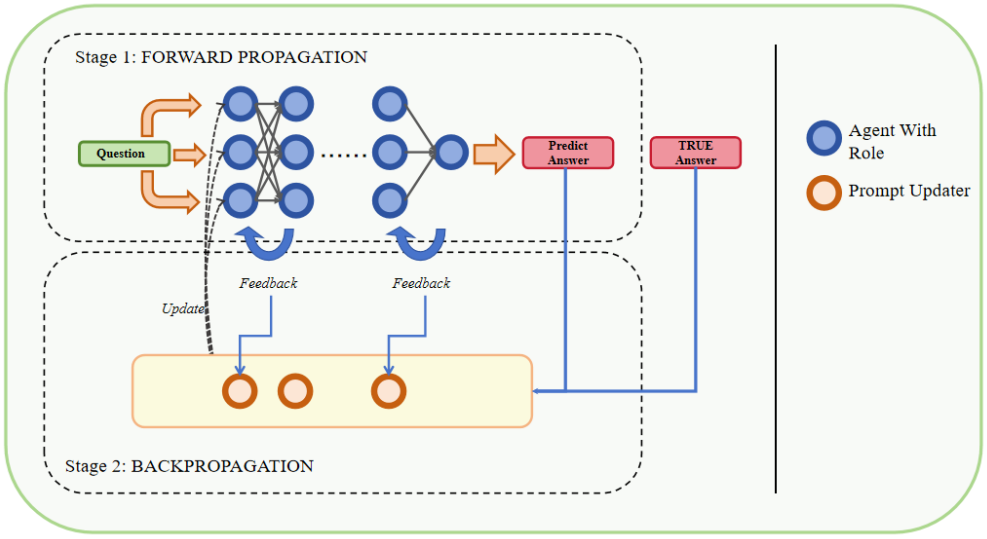

Figure 2: The architecture and optimization mechanism of AgenticNet. The framework operates through a dual-stage process mirroring neural network training. In the Forward Propagation (Stage 1), the input query traverses a layered network where agents function as computational units. Agents actively process upstream information by applying preference ranking and Top-K context selection to filter relevant signals before generating intermediate reasoning steps. When the final prediction diverges from the ground truth, the Textual Backpropagation (Stage 2) is initiated. Here, a task-specific loss acts as an error signal that activates Layer Prompt Updaters. These updaters analyze the discrepancy alongside the forward pass trace to synthesize textual gradients—natural language feedback specifying how to correct the reasoning logic. Finally, a Rewrite operator applies these gradient estimates to iteratively refine the agents' system prompts.

DyLAN Liu et al. (2024) builds a dynamic agent network that selects and configures a team of agents per task, rather than using fixed agents, to better match task needs. Other systems evolve their structure as the task progresses: EvoFlow Zhang et al. (2025a) uses heuristic search to adjust the workflow "on the fly," refining the sequence of agent actions when initial plans fail. On the other hand, works like GPTSwarm and AFlow adapt the coordination topology to improve outcomes Zhuge et al. (2024); Zhang et al. (2024); Hu et al. (2024b); Shang et al. (2024); Zhang et al. (2025a). In our work, we propose a layered architecture with fixed roles and a loss-driven update for adapting tasks while improving the agents ability in tailored tasks, aiming for a design that is easier to scale and interpret.

3 Methodology

We present AgenticNet, a multilayer multi-agent architecture that mirrors the computational structure of a multilayer perceptron (MLP) while operating in text space. AgenticNet formalizes agents as nodes in a MLP whose parameters are natural-language prompts; layers of such agents compose into a feed-forward computation graph and are trained by textual gradient. Below we describe the architecture and the learning procedure.

3.1 Architecture

Let q denote an input query and $y^\star$ the ground-truth answer. AgenticNet is composed of L layers indexed by $l = 1, \ldots, L$; layer l contains $N^{(l)}$ agents. The prompt of the i-th agent in layer l is $p_i^{(l)}$. The model before training is $f_\theta(\cdot)$. We denote the set of textual outputs produced by layer l as

$$X^{(l)} = \{x_i^{(l)}\}_{i=1}^{N^{(l)}}, \tag{1}$$

where each $x_i^{(l)}$ is output of agent index i in layer l.

The topmost layer contains a dedicated Synthesizer agent with prompt $p^{(\text{final})}$ that consumes $X^{(L)}$ and emits a single consolidated prediction $\hat{y}$. A judge returns a dataset-dependent vector of diagnostic signals

$$\mathbf{L} = (L_1, \ldots, L_m), \qquad L_t \in \mathcal{C}_t, \tag{2}$$

where m and the semantics of each component L_t (e.g. reasoning validity, arithmetic correctness, answer correctness, format compliance) are chosen according to the target task. In the simplest case $\mathcal{C}_t = \{0, 1\}$. These signals are combined into a scalar training objective by a weighted sum,

$$\mathcal{L}(\hat{y}, y^\star) = \sum_{t=1}^{m} w_t \big(1 - \ell_t(\hat{y}, y^\star)\big), \tag{3}$$

where $\ell_t(\hat{y}, y^\star)$ denotes the judge-provided score (equal to L_t for binary criteria) and $w_t \geq 0$ are validation-selected weights.

3.2 Forward propagation

We denote the input wrapper as $X^{(0)} = \{Question : q\}$. For each layer $l = 1, \ldots, L$ and for each agent $i \in \{1, \ldots, N^{(l)}\}$, the forward computation proceeds by letting the agent rank all upstream messages $\{x_j^{(l-1)}\}_{j=1}^{N^{(l-1)}}$ according to its own preference. A compact context is then constructed by selecting the top-k messages in the ranked list,

$$\mathcal{J}_i^{(l)} = \text{TopK}\left(\{x_j^{(l-1)}\}_{j=1}^{N^{(l-1)}}, k\right), \qquad \tilde{x}_i^{(l)} = x_{j_1}^{(l-1)} \oplus x_{j_2}^{(l-1)} \cdots \oplus x_{j_k}^{(l-1)} \tag{4}$$

where indices $j_1, \ldots, j_k$ follow the descending order in the agent's ranking and $\oplus$ denotes textual concatenation under a fixed prompt-template that specifies the agent's role. The agent then generates its output conditioned on this context as

$$x_i^{(l)} = f_\theta\big(p_i^{(l)} \oplus \tilde{x}_i^{(l)} \oplus q\big) \tag{5}$$

The Synthesizer agent collates and analyze $X^{(L)}$ under $p^{(\text{Synthesizer})}$ then directly generates a consolidated $\hat{y}$.

3.3 Backward propagation

The backward propagation in AgenticNet is not the numeric gradient backpropagation of standard neural networks; instead, adaptation is driven by textual gradient estimates produced by large language models. The core idea is to use the scalar loss $\mathcal{L}$ provided by the judge, together with the layer-wise intermediate outputs, to generate natural-language modification suggestions for each agent prompt.

Concretely, at layer l, a Layer Prompt Updater agent inspects the current prompt set

$$P^{(l)} = \{p_i^{(l)}\}_{i=1}^{N^{(l)}}, \tag{6}$$

the upstream and downstream outputs $(X^{(l-1)}, X^{(l)})$, the ground-truth answer $y^\star$, and the overall loss $\mathcal{L}$. For each input-output pair $(x_j^{(l-1)}, x_j^{(l)})$, the Layer Prompt Updater generates a feedback message

$$f_j^{(l)} = F\big(x_j^{(l-1)}, x_j^{(l)}, y^\star, \mathcal{L}\big), \tag{7}$$

which provides actionable suggestions for improving the corresponding agent's prompt. All feedback messages are collected into a set

$$\mathcal{F}^{(l)} = \{f_j^{(l)}\}_{j=1}^{|X^{(l)}|}. \tag{8}$$

Based on this set of feedback, the Layer Prompt Updater produces for each agent i a textual modification suggestion

$$\Delta p_i^{(l)} = \mathrm{Aggregate}\big(\mathcal{F}^{(l)}\big), \tag{9}$$

which we interpret as a textual–gradient estimate, denoted $\widehat{\nabla}_{p_i^{(l)}}\mathcal{L}$, indicating how $p_i^{(l)}$ should be changed to reduce $\mathcal{L}$. For analogy with gradient descent, we write the conceptual update as

$$p_i^{(l)} \leftarrow p_i^{(l)} - \widehat{\nabla}_{p_i^{(l)}}\mathcal{L}, \tag{10}$$

where the subtraction symbol denotes applying the updater agents' suggested modification. It is important to emphasize that $\widehat{\nabla}_{p_i^{(l)}}\mathcal{L}$ is not a true numerical vector but an abstract representation of a language-formatted update.

In practice, the prompt update is carried out by a Rewrite step: the current prompt and the suggested modifications are fed to a rewrite procedure that produces a candidate new prompt

$$p_i^{(l)\,\mathrm{new}} = \mathrm{Rewrite}\big(p_i^{(l)}, \Delta p_i^{(l)}\big). \tag{11}$$

Besides, a Global Prompt Updater is introduced to update the prompt of layer prompt updaters and ensure that layer-specific objectives are aligned with the overall task. Concretely, the Global Prompt Supervisor receives the prompt of the Layer Prompt Updater for layer l, $P^{(l)}$, along with the corresponding inputs and outputs of that layer, $X^{(l-1)}$ and $X^{(l)}$. Based on this information and the total loss $\mathcal{L}$, it generates a textual modification suggestion$\Delta P^{(l)}$, where $\Delta P^{(l)}$ represents an abstract, language-formatted update direction to improve the overall performance of the layer. The final updated prompt for the layer is obtained by combining the local suggestions $\Delta P^{(l)}$ from the Layer Prompt Updater with the global guidance:

$$P^{(l)\,\mathrm{new}} = \mathrm{Rewrite}\Big(P^{(l)}, \Delta P^{(l)}_{\mathrm{global}}\Big) \tag{12}$$

where Rewrite applies the textual modifications to produce a new candidate prompt. Through this mechanism, the Global Prompt Updater provides cross-agent and cross-output coordination within the layer.

4 Experiment

4.1 Experiment Setup

Benchmarks. We evaluate `AgenticNet` on five challenging public benchmarks that span the domains of mathematical reasoning and code generation. These include: **GSM8K** Cobbe et al. (2021), a dataset of grade-school math word problems; **MATH** Hendrycks et al. (2021), which contains more difficult high-school competition-level math problems; **AIME2024** HuggingFaceH4 (2024), a set of challenging math competition problems; **HumanEval** Chen et al. (2021) for evaluating code generation capabilities; and **MBPP** Austin et al. (2021) for Python code generation. For MATH dataset, we selected a subset of 500 problems at difficulty level 5 for the main comparison against other MAS.

Baselines. To provide a comprehensive evaluation, we compare our framework against 14 state-of-the-art methods, which we categorize into three groups:

- **Single-Agent Methods:** These include standard prompting techniques such as CoT Wei et al. (2022), ComplexCoT Fu et al. (2023), and Self-Consistency (SC) Wang et al. (2023).
- **Hand-Crafted Multi-Agent Systems:** These systems rely on manually defined roles and workflows, including MultiPersona Wang et al. (2024), LLM-Debate Du et al. (2023), LLM-Blender Jiang et al. (2023), AgentVerse Chen et al. (2023a), and MacNet Qian et al. (2025).
- **Automated Multi-Agent Systems:** This category covers frameworks that dynamically search for or adapt agent workflows, including DyLAN Liu et al. (2024), AutoAgents Chen et al. (2024), GPTSwarm Zhuge et al. (2024), ADAS Hu et al. (2024a), AgentSquare Shang et al. (2024), and AFlow Zhang et al. (2024).

Table 1: **Performance comparison** among single-agent, hand-crafted, and automated multi-agent systems. All baselines use `gpt-4o-mini` as the base model. Best and second-best results are **bolded** and underlined, respectively.

Method	GSM8K	MATH	HumanEval	MBPP	Avg.
Vanilla	87.45	46.29	87.08	71.83	73.16
CoT Wei et al. (2022)	87.10	46.40	88.13	71.83	73.87
ComplexCoT Fu et al. (2023)	86.89	46.53	87.49	72.36	73.32
SC (CoT×5) Wang et al. (2023)	87.57	47.91	88.60	73.60	74.42
MultiPersona Wang et al. (2024)	87.50	45.43	88.32	73.19	73.61
LLM-Debate Du et al. (2023)	89.47	48.54	88.68	70.29	74.25
LLM-Blender Jiang et al. (2023)	88.35	46.92	88.80	77.05	75.28
DyLAN Liu et al. (2024)	89.98	48.63	90.42	77.30	76.58
AgentVerse Chen et al. (2023a)	89.91	47.35	89.29	74.28	75.21
MacNet Qian et al. (2025)	87.95	45.18	84.57	65.28	70.75
AutoAgents Chen et al. (2024)	87.69	45.32	87.64	71.95	73.15
GPTSwarm Zhuge et al. (2024)	89.14	47.88	89.32	77.43	75.94
ADAS Hu et al. (2024a)	86.12	43.18	84.19	68.13	70.41
AgentSquare Shang et al. (2024)	87.62	48.51	89.08	78.46	75.92
AFlow Zhang et al. (2024)	91.16	51.28	90.93	81.67	78.76
MaAS Zhang et al. (2025c)	**92.30**	51.82	92.85	82.17	82.25
MermaidFlow Zheng et al. (2025)	91.4	**55.42**	92.87	82.31	82.77
AgenticNet (Ours)	91.8	49.03	**98.8**	**98.7**	**85.56**

Implementation Details. For a fair and direct comparison, all baseline results reported in Table 1 are based on the `gpt-4o-mini` model. Our cross-model evaluation in Table 3 also includes results using `GPT-4.1` and `DeepSeek-V3.1` to demonstrate the generalizability of our framework. Our `AgenticNet` is configured with $L = 3$ layers. Specifically, the first layer contains $N^{(1)} = 2$ agents, while the subsequent layers contain $N^{(2)} = N^{(3)} = 3$ agents. The textual backpropagation mechanism is applied after each task to update agent prompts based on the feedback from the next layer.

4.2 Performance Analysis

As evidenced by Table 1, `AgenticNet` consistently outperforms both single-agent and multi-agent baselines. Utilizing `gpt-4o-mini` as the common base model, our framework achieves an average score of 85.56%. Notably, in code generation tasks (HumanEval and MBPP), `AgenticNet` achieves a substantial lead, reaching 98.8% and 98.7% respectively.

4.3 Ablation Study

To isolate the impact of the textual backpropagation mechanism, we conducted an ablation study where the prompt update process was disabled, referred to as "w/o Training." This comparison allows us to highlight the advantages of our proposed learning procedure, separate from the structural benefits provided by the multi-agent layout.

As demonstrated in Table 2, the training mechanism plays a crucial role in enhancing performance across complex domains. For instance, in the MATH and AIME benchmarks, transitioning from a zero-shot configuration to an optimized one leads to substantial improvements. In the AIME benchmark, for example, accuracy increases from 43.3% to 50.0%. Notably, while the structural benefits alone establish a strong baseline in coding tasks, such as the 98.17% accuracy on HumanEval, the iterative refinement process further

Table 2: Ablation study on the training mechanism. The experiment is conducted using different base models to show the impact of textual backpropagation.

Benchmark	w/o Training	with Training
MATH	96.4%	**83.2%**
AIME	43.3%	**50.0%**
HumanEval	98.17%	**100%**
MBPP	100%	**97.2%**

Table 3: Performance comparison across benchmarks with Vanilla and AgenticNet methods. Results are shown for three base models: GPT-4o-mini, GPT-4.1, and DeepSeek-V3.1.

Model	Benchmark	Vanilla	AgenticNet
GPT-4o-mini	GSM 8K	87.4	91.8
	MATH 500	70.2	71.6
	AIME 2024	11.5	6.7
	Human Eval	87.0	91.4
	MBPP	85.4	88.9
GPT-4.1	GSM 8K	90.0	92.2
	MATH 500	87.2	89.4
	AIME 2024	39.6	50.0
	Human Eval	95.6	98.8
	MBPP	88.0	98.7
DeepSeek-V3.1	GSM 8K	89.3	93.5
	MATH 500	91.5	83.2
	AIME 2024	52.2	50.0
	Human Eval	98.2	100.0
	MBPP	61.1	97.2

enhances performance, ultimately reaching near-perfect results of 100%. This highlights the significance of iterative prompt optimization in addressing edge-case logical errors.

4.4 Transferability and Model-Agnostic Analysis

A robust multi-agent framework must exhibit generalization across different model architectures. We evaluated the model-agnostic nature of `AgenticNet` by applying the same optimized framework to diverse LLM backbones: `GPT-4o-mini`, `GPT-4.1`, and `DeepSeek-V3.1`.

As detailed in Table 3, `AgenticNet` provides consistent performance enhancements regardless of the underlying model. When integrated with `GPT-4.1`, the framework improves AIME 2024 performance from 39.6% to 50.0%, representing a 10.4 percentage point absolute gain. The generalizability is even more pronounced with `DeepSeek-V3.1`, where MBPP scores surged from 61.1% to 97.2%—a remarkable 36.1 point increase. Even for smaller models like `GPT-4o-mini`, the framework yields significant improvements in GSM8K (from 87.4% to 91.8%). These consistent gains across disparate architectures suggest that the reasoning patterns and cooperative strategies learned via textual backpropagation are model-agnostic, effectively capturing high-level logic that transcends specific model weights.

4.5 Impact of Network Depth

To investigate the scalability of AgenticNet and empirically identify the optimal architectural depth, we extended our evaluation on the MATH dataset by varying the number of layers $L \in \{2, 3, 4, 5\}$. This analysis was performed using two distinct base models which are `GPT-4o-mini` and `DeepSeek-V3` to explore how model capacity interacts with network depth.

As shown in Figure 3 and summarized in Table 4. the results reveal notable differences in how depth impacts performance depending on the model's intrinsic capabilities. For the highly capable `DeepSeek-V3`,

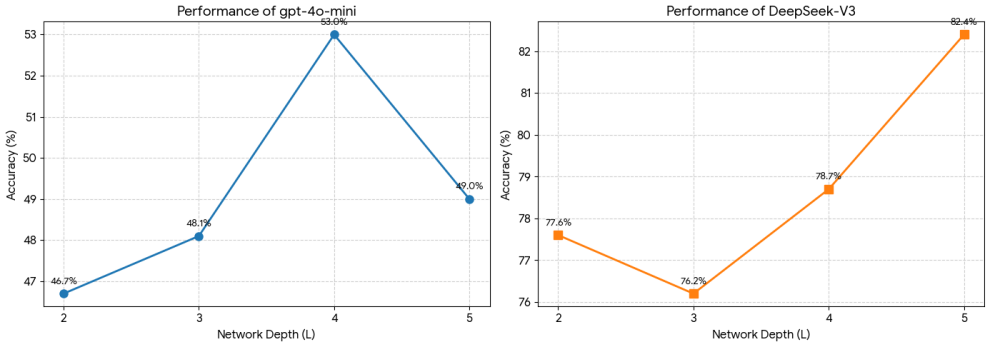

Figure 3: Performance of AgenticNet with `gpt-4o-mini` and `Deepseek-V3`backbone across different network depths.

Table 4: Performance comparison on the MATH dataset with varying network depths (L). Accuracy is reported in percentage (%).

Base Model	$L = 2$	$L = 3$	$L = 4$	$L = 5$
`gpt-4o-mini`	46.7	48.1	**53.0**	49.0
`DeepSeek-V3`	77.6	76.2	78.7	**82.4**

performance shows a clear positive correlation with depth within the tested range. Accuracy improves from 77.6% at $L = 2$ to a peak of 82.4% at $L = 5$. This trend suggests that more powerful base models have the reasoning stability necessary to fully exploit deeper architectures.

In contrast, `GPT-4o-mini` exhibits a more complex, nonlinear performance trajectory. While accuracy increases from 46.7% at $L = 2$ to a maximum of 53.0% at $L = 4$, further increasing the depth to $L = 5$ results in a performance drop to 49.0%.

4.6 Training Dynamics and Convergence Analysis

Table 5: Evolution of performance on the MATH dataset across training epochs using the `GPT-4o-mini` backbone. Accuracy is reported in percentage (%).

Metric	**Epoch 1**	**Epoch 2**	**Epoch 3**	**Epoch 4**	**Epoch 5**
Accuracy (%)	49.0	51.9	**52.7**	52.1	52.5

To empirically validate the efficacy and stability of the textual backpropagation mechanism, we examine the optimization trajectory of the agent across sequential training iterations. In this experimental setup, an "epoch" is defined as a complete optimization cycle, which includes the generation of textual gradients based on task-specific loss and the subsequent application of the `Rewrite` operator to refine the system prompts. Using the `gpt-4o-mini` backbone, we track the evolution of reasoning accuracy on the MATH dataset to assess how iterative prompt updates contribute to performance improvements.

As shown in Figure 4 and summarized in Table 5, the training dynamics follow a pattern of rapid initial adaptation followed by stable convergence, in contrast to the volatility often observed in discrete optimization approaches. Starting from a zero-shot baseline of 49.0% at Epoch 1, the system demonstrates a steep learning curve during the early phase. Accuracy improves significantly to 51.9% by Epoch 2 and peaks at 52.7% in Epoch 3. This consistent growth during the initial stages suggests that the generated textual gradients are highly effective in identifying and addressing key reasoning deficiencies within the initial prompt distribution.

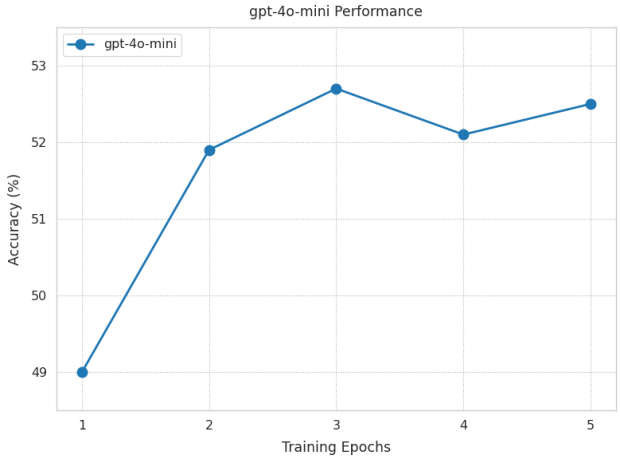

Figure 4: Learning trajectory of AgenticNet with `GPT-4o-mini` over 5 training epochs. The curve illustrates a rapid optimization phase followed by stable convergence, peaking at Epoch 3.

Table 6: Performance scaling of AgenticNet relative to network width W. Results are reported for the MATH reasoning benchmark.

Width (W)	**Accuracy (%)**
$W = 1$	49.0
$W = 2$	48.3
$W = 3$	49.0
$W = 4$	**50.9**

After this rapid optimization phase (Epochs 4 and 5), the performance enters a phase of asymptotic stability. Although minor fluctuations are observed, with accuracy reaching 52.1% and 52.5%, respectively, the overall accuracy remains robust compared to the baseline, with an average gain of +3.5%. This plateau suggests that the agent has converged to a local optimum within the semantic search space. The slight variations in the final epochs are attributed to the inherent stochasticity of large language model generation rather than any instability in the optimization algorithm, reinforcing the conclusion that the learned prompts are stable and resistant to degradation over extended training steps.

4.7 Impact of Network Width

To evaluate the horizontal scalability of AgenticNet, we investigate the impact of network width W on reasoning performance. In this context, width W is defined as the number of parallel agents within the Executor and Synthesizer layers. We conduct a series of experiments on the MATH dataset using the `gpt-4o-mini` backbone, varying $W \in \{1, 2, 3, 4\}$ while maintaining a constant first layer.

As shown in Table 6, the system demonstrates a unique performance trajectory as the width increases. At $W = 1$ and $W = 3$, the architecture achieves a stable accuracy of 49.0%. Interestingly, a slight performance dip to 48.3% is observed at $W = 2$, suggesting a potential transition phase where increased agent interaction introduces minor noise before achieving collective synergy. However, when the width is further expanded to $W = 4$, AgenticNet reaches its peak performance with an accuracy of 50.9%.

These results suggest that AgenticNet exhibits a scaling behavior in which increasing the number of parallel agents enhances the system's capacity to evaluate multiple reasoning paths and integrate diverse solutions. The notable performance improvement at $W = 4$ indicates that a wider architecture provides greater robustness, enabling the system to successfully navigate complex logical challenges that single-agent or narrower configurations may fail to resolve.

4.8 Case Study

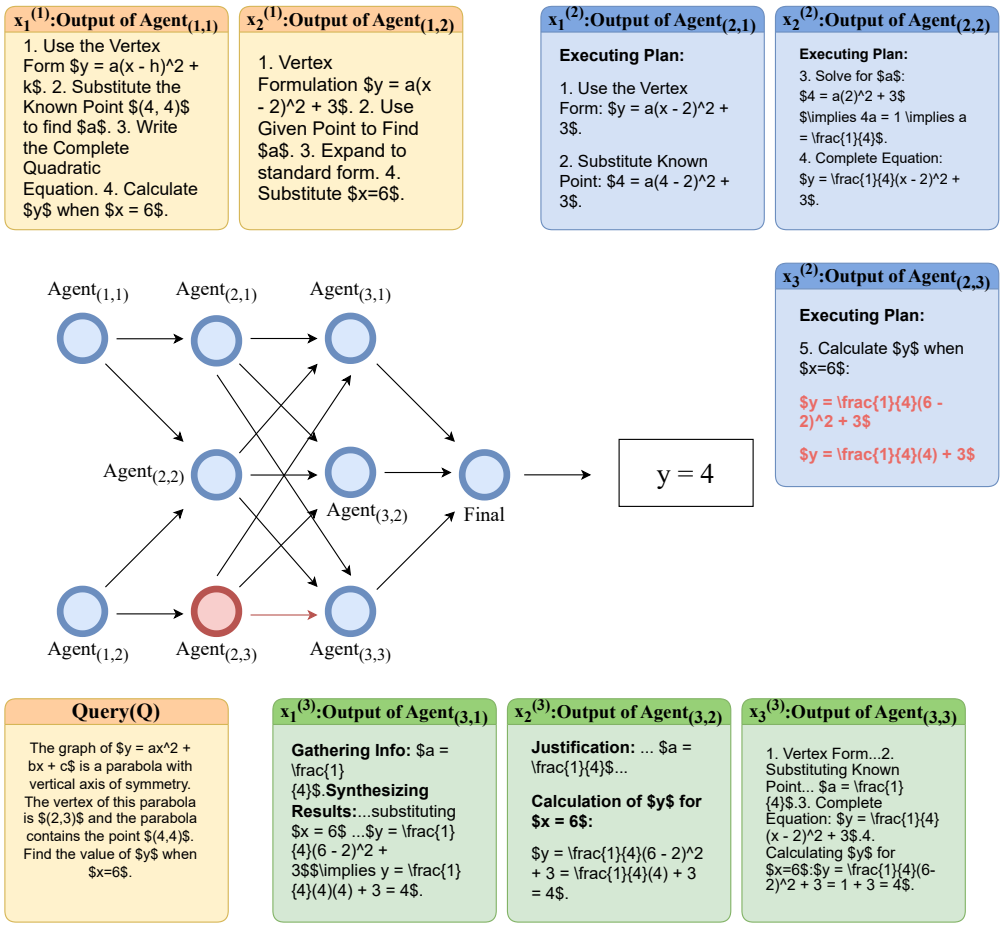

Figure 5: The Forward Propagation stage for the case study. The query q is processed by layer 1 and layer 2. A calculation error (highlighted in red) originates in agent $x_3^{(2)}$, which propagates through the layer 3 to produce an incorrect final answer $\hat{y} = 4$.

To demonstrate the end-to-end operational flow and adaptive learning capabilities of AgenticNet, we present a qualitative case study from the MATH reasoning benchmark. This example illustrates how the network processes a complex query, the propagation of a subtle arithmetic error through layers, and the mechanism by which textual backpropagation identifies and rectifies the underlying logic.

4.8.1 Problem Definition

Consider the input query q from the MATH reasoning benchmark: "The graph of $y = ax^2 + bx + c$ is a parabola with a vertical axis of symmetry. The vertex of this parabola is $(2, 3)$ and the parabola contains the point $(4, 4)$. Find the value of y when $x = 6$." The ground-truth answer for this problem is $y^* = 7$.

4.8.2 Forward Propagation and Error Manifestation

In the forward propagation pass (see Figure 5), the query is processed by a 3-layer network. In Layer 1 , the system generates a coherent multi-step decomposition, denoted as $X^{(1)}$, which is then passed to Layer 2. The initial executor agents correctly derive the vertex form of the equation $y = a(x - 2)^2 + 3$ and solve for the coefficient $a = \frac{1}{4}$.

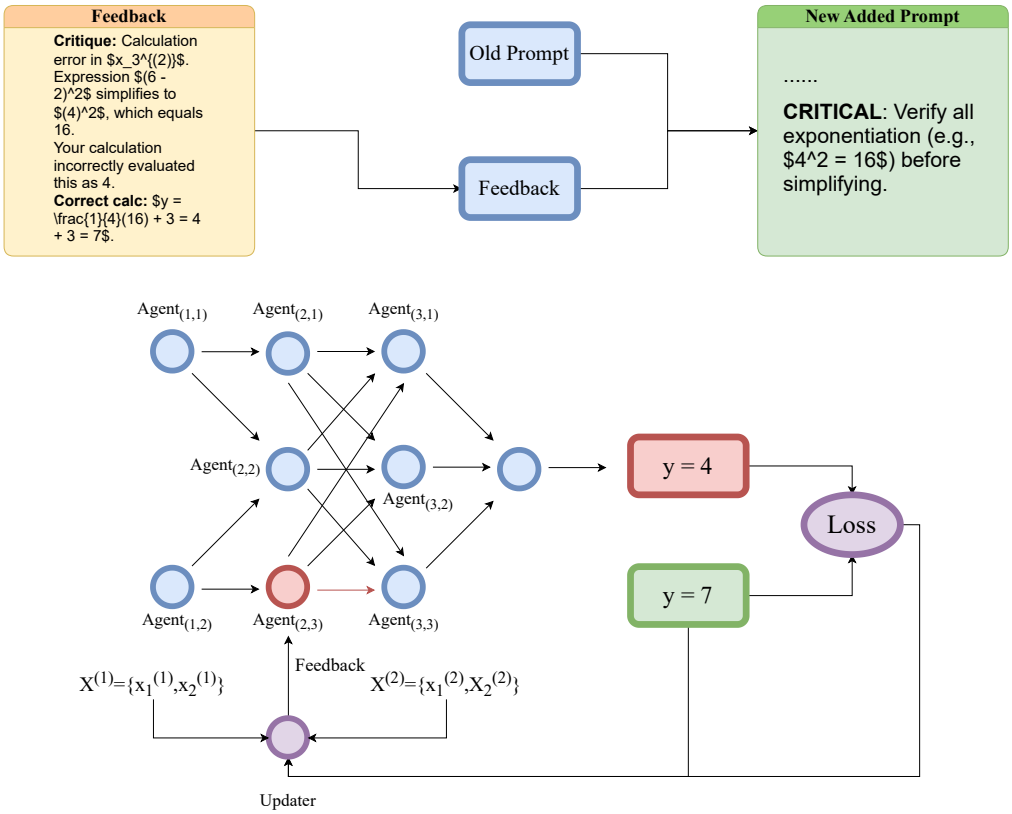

Figure 6: The Textual Backpropagation stage for the case study. The non-zero loss $\mathcal{L}$ activates the Layer Prompt Updater (LPU). The LPU inspects the forward pass data (dashed lines), identifies the error source ($x_3^{(2)}$), and generates a specific feedback message ($f_3^{(2)}$). This "textual gradient" is used by the `Rewrite` procedure to update the agent's prompt $p_3^{(2)}$ to $p_3^{(2)new}$, correcting the logic for future tasks.

However, in Layer 2, agent $x_3^{(2)}$, which is responsible for the final numerical evaluation, introduces a critical arithmetic oversight. The agent incorrectly evaluates 4^2 as 4 instead of the correct value 16. This flawed intermediate result propagates to Layer 3 , where the error goes undetected, leading to an incorrect final prediction $\hat{y} = 4$.

4.8.3 Textual Backpropagation and Optimization

The textual backpropagation stage (Figure 6) is triggered when the Judge component identifies a discrepancy between the predicted output $\hat{y} = 4$ and the ground-truth value $y^* = 7$. This non-zero task loss $\mathcal{L}$ signals a reasoning failure and activates the Layer Prompt Updater (LPU) for Layer 2.

The LPU inspects the forward pass data $(X^{(1)}, X^{(2)})$ alongside the ground truth, identifying the error source in agent $x_3^{(2)}$. It then generates a targeted feedback message $f_3^{(2)}$, which serves as a textual gradient estimate $\hat{\nabla}_{p_3^{(2)}}\mathcal{L}$:"The execution of Step 5 is incorrect. The expression $(6-2)^2$ simplifies to 4^2, which equals 16. Your calculation incorrectly evaluated this as 4. The correct calculation is $y = \frac{1}{4}(16) + 3 = 7$."

This feedback is processed by the `Rewrite` procedure, which generates a modification $\Delta p_3^{(2)}$ to update the agent's prompt $p_3^{(2)}$. The updated prompt includes a new preventative rule: *When evaluating mathematical expressions, meticulously verify all exponentiation operations (e.g., verify* $(4)^2$ *is 16) before simplifying.*

5 Conclusion

In this paper, we introduced AgenticNet, a novel multi-agent framework inspired by neural networks to address the ad-hoc nature of many MAS designs. Unlike prevailing methods that rely on dynamic search, AgenticNet utilizes a fixed, layered architecture with specialized roles (Planners, Executors, and Synthesizers). We proposed a textual backpropagation mechanism, driven by task loss, to "train" the agents' prompts, enabling systematic and structured adaptation. Extensive experiments demonstrated that AgenticNet achieves state-of-the-art average performance, driven by dominant results on code generation tasks. Ablation studies further confirmed that our loss-driven training mechanism is a critical component for success in complex reasoning. In summary, AgenticNet offers a principled, scalable, and interpretable new direction for multi-agent system design, paving the way for future explorations into more complex network topologies and broader applications.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation, 2024. URL https://arxiv.org/abs/2309.17288.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors, 2023a. URL https://arxiv.org/abs/2308.10848.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2(4):6, 2023b.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. In *arXiv preprint arXiv:2110.14168v2*, 2021. Version v1 submitted on 27 Oct 2021, v2 (this version) revised 18 Nov 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning, 2023. URL https://arxiv.org/abs/2210.00720.

Alireza Ghafarollahi and Markus J Buehler. Protagents: protein discovery via large language model multi-agent collaborations combining physics and machine learning. *Digital Discovery*, 3(7):1389–1409, 2024.

Alireza Ghafarollahi and Markus J Buehler. Sciagents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024a. URL https://arxiv.org/abs/2308.00352.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. International Conference on Learning Representations, ICLR, 2024b.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024a.

Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. *arXiv preprint arXiv:2410.16946*, 2024b.

Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

HuggingFaceH4. Aime_2024 dataset. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2024. Accessed: 2025-10-30.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion, 2023. URL https://arxiv.org/abs/2306.02561.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023a.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023b. URL https://arxiv.org/abs/2303.17760.

Nian Li, Chen Gao, Mingyu Li, Yong Li, and Qingmin Liao. Econagent: large language model-empowered agents for simulating macroeconomic activities. *arXiv preprint arXiv:2310.10436*, 2023c.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL https://arxiv.org/abs/2304.03442.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration, 2025. URL https://arxiv.org/abs/2406.07155.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. *arXiv preprint arXiv:2410.06153*, 2024.

Peiran Wang, Yaoning Yu, Ke Chen, Xianyang Zhan, and Haohan Wang. Large language model-based data science agent: A survey. *arXiv preprint arXiv:2508.02744*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration, 2024. URL https://arxiv.org/abs/2307.05300.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. *arXiv preprint arXiv:2406.14228*, 2024.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*, 2025a.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025b.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025c. URL https://arxiv.org/abs/2502.04180.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

Qinlin Zhao, Jindong Wang, Yixuan Zhang, Yiqiao Jin, Kaijie Zhu, Hao Chen, and Xing Xie. Competeai: Understanding the competition dynamics in large language model-based agents. *arXiv preprint arXiv:2310.17512*, 2023.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming, 2025. URL https://arxiv.org/abs/2505.22967.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.