Self-Evolving Multi-Agent Systems via Textual Backpropagation

Anonymous ACL submission

Abstract

Leveraging multiple Large Language Models (LLMs) has proven effective for addressing complex, high-dimensional tasks, but current approaches often rely on static, manually engineered multi-agent configurations. To overcome these constraints, we present the Agentic Neural Network ($\mathcal{ANN}$), a framework that conceptualizes multi-agent collaboration as a layered neural network architecture. In this design, each agent operates as a node, and each layer forms a cooperative team focused on a specific subtask. Agentic Neural Network follows a two-phase optimization strategy: (1) Forward Phase - Drawing inspiration from neural network forward passes, tasks are dynamically decomposed into subtasks, and cooperative agent teams with suitable aggregation methods are constructed layer by layer. (2) Backward Phase - Mirroring backpropagation, we refine both global and local collaboration through iterative feedback, allowing agents to self-evolve their roles, prompts, and coordination. This neuro-symbolic approach enables $\mathcal{ANN}$ to create new or specialized agent teams post-training, delivering notable gains in accuracy and adaptability. Across seven benchmark datasets, $\mathcal{ANN}$ surpasses leading multi-agent baselines under the same configurations, showing consistent performance improvements.

1 Introduction

Large Language Models (LLMs) have ushered in a new era of artificial intelligence, exhibiting strong capabilities in reasoning, content generation, and multi-step problem-solving (Kojima et al., 2022; Ouyang et al., 2022). By grouping these models into *multi-agent systems* (MAS), researchers have addressed an array of complex tasks, ranging from code generation and debugging (Jimenez et al., 2023) to retrieval-augmented generation (Khattab et al., 2023a; Lewis et al., 2020; Gao et al., 2023) and data analysis (Hong et al., 2024; Hu et al., 2024). Often, MAS outperform their single-agent equivalents by bringing together diverse agent roles and expertise, including verifier agents (Shinn et al., 2023) or debating agents (Qian et al., 2024; Zhuge et al., 2024b), thus creating more adaptable and robust solutions. However, designing and deploying effective MAS remains demanding. Developers frequently invest substantial effort into prompt engineering, role assignment, and topology definition by trial and error (Chen et al., 2023; Hong et al., 2023), especially for dynamic, high-dimensional tasks.

Recent advances in automating aspects of MAS design aim to relieve these challenges. For instance, Khattab et al. (2024) introduced systematic methods for generating in-context exemplars; Hu et al. (2025) presented a meta-agent capable of creating new topologies in code; and Zhang et al. (2024) employed Monte Carlo Tree Search to find improved workflow configurations; Ke et al. (2025) proposed an automatic MAS optimization architecture under zero supervision and demonstrated significant gains. These innovations mirror earlier developments in MAS design research, where layer-wise optimization gave way to holistic, end-to-end backpropagation (Jacobs et al., 1991; Hinton et al., 2006). Similarly, *symbolic* or *agent-level* frameworks that model entire multi-agent pipelines as computational graphs have emerged (Khattab et al., 2023a; Zhuge et al., 2024a; Zhou et al., 2024).

Building on these insights, we introduce the *Agentic Neural Network ($\mathcal{ANN}$)*, a framework that adapts principles from classic neural networks to orchestrate multiple LLM agents. As shown in Figure 1, conventional neural networks rely on learnable weights and numeric optimizers for end-to-end training via gradient-based updates, whereas $\mathcal{ANN}$ considers each layer as a team of language agents, jointly optimizing roles, prompts, and tools through textual gradients (Yuksekgonul et al., 2024). While Mixture of Experts (MoE)

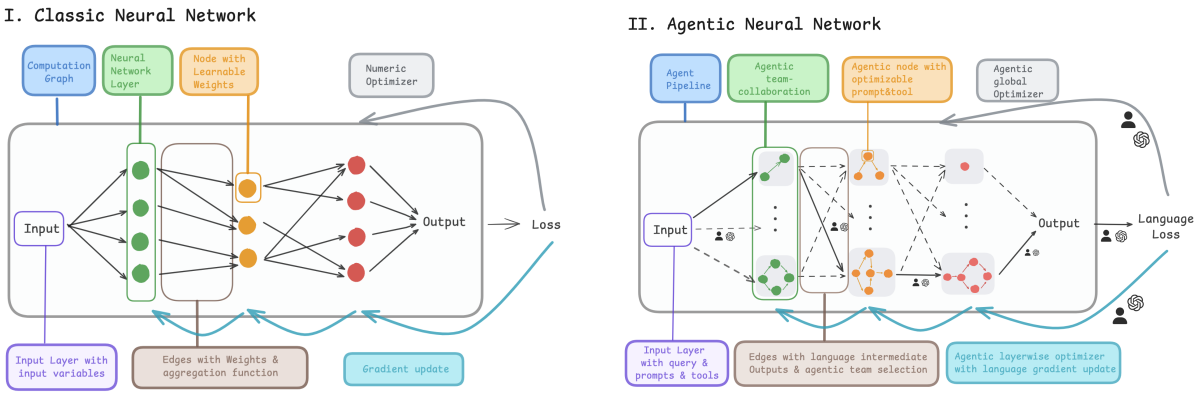

Figure 1: A conceptual comparison between classic neural networks (left) and our $\mathcal{ANN}$ (right). In the right-hand agentic diagram, the brown module labeled *"Edges with language intermediate Outputs & agentic team selection"* represents the choice among multiple candidate collaboration strategies between agent teams. **Solid lines** indicate selected collaboration modes that form the pipeline connection between layers, while **dashed lines** represent alternative strategies that were not selected at that step.

and Mixture of Agents (MoA) architectures aim to scale model capacity through gated expert selection within a monolithic model (Shazeer et al., 2017; Wang et al., 2024b), $\mathcal{ANN}$ organizes layerwise teams of language agents that collaborate through multi-step reasoning and are refined via textual gradients (Yuksekgonul et al., 2024). This design enables $\mathcal{ANN}$ to support flexible, role-based agent coordination beyond the scope of numeric expert gating.

Instead of a purely engineering-driven approach, $\mathcal{ANN}$ divides a complex task into smaller subproblems, assigning each to a layer of specialized agents, and iteratively refines both local design (i.e., agent prompts and configurations) and global coordination (i.e., inter-layer flows and topologies). Our approach proceeds in two stages. First, during the forward agent team generation phase, the main task is decomposed into subtasks, with specialized agent teams dynamically assigned layer by layer, ensuring each layer is responsible for a distinct subtask. Then, during training, if performance is suboptimal, the backward agent team optimization phase backpropagates textual feedback to isolate errors and propose targeted adjustments. These textual critiques act like gradient signals, guiding prompt updates and connection refinements (Yao et al., 2022; Verma, 2024; Khattab et al., 2023a).

To illustrate the capabilities of our framework, we evaluate $\mathcal{ANN}$ on four challenging datasets: MATH (Hendrycks et al., 2021) for mathematical reasoning, DABench (Hu et al., 2024) for data analysis, Creative Writing (Zhou et al., 2024) for open-ended writing tasks, HumanEval (Chen et al., 2021) for code generation, and MMLU (Hendrycks et al., 2020) for multiple-choice question answering. Our experiments show that our framework not only simplifies MAS design by automating prompt tuning, role assignment, and agent collaboration but also outperforms existing baselines in accuracy. Through this process, the framework acquires self-evolving capabilities, dynamically reconfiguring its agent teams and coordination strategies to meet the demands of novel tasks.

2 Related Works

In this section, we review the evolution of AI agents into LLM-based systems, discuss the emerging concept of agentic workflows, survey automated methods for optimizing agent configurations, and outline the remaining challenges in multi-agent settings.

Evolution of AI Agents Early AI agents were highly specialized and depended chiefly on symbolic reasoning, as seen in board-game-playing systems like Chess and Go. Subsequent innovations introduced reactive and reinforcement learning agents with greater adaptability. More recently, LLM-based agents have appeared, incorporating large-scale language models (Radford et al., 2018, 2019; Ouyang et al., 2022) at their foundation. By processing natural language inputs and outputs, these agents enable more flexible, human-like interactions and reasoning.

LLM-Based Agentic Workflows Modern workflows often rely on multiple LLM invocations to address complex, multi-step tasks (Wei et al., 2022;

Madaan et al., 2023; Gao et al., 2022). In these agentic workflows, each stage or node corresponds to specific subtasks like prompt creation, tool utilization, or domain-specific strategies (Hong et al., 2023; Yang et al., 2023; Cai et al., 2023). Through specialized roles—including data analyzers, verifiers, or debaters—LLM-based agents can collaborate efficiently on a range of domain challenges, from code generation (Hong et al., 2024; Lee et al., 2023) to advanced data analysis (Li et al., 2024a).

Automated Optimization Approaches As task workflows grow more involved, automated methods aim to minimize manual engineering. *Prompt optimization* tailors textual inputs to steer LLM outputs (Khattab et al., 2023a; Zhuge et al., 2024b). *Hyperparameter tuning* fine-tunes model parameters or scheduling (Liu et al., 2024a), and *workflow optimization* revises entire computational graphs or code structures (Hu et al., 2025; Zhang et al., 2024; Zhuge et al., 2024a). Symbolic learning frameworks (Hong et al., 2024; Zhuge et al., 2024b; Zhou et al., 2024) optimize prompts, tools, and node configurations collectively, mitigating local optima that can emerge from optimizing each component independently. Furthermore, Lee et al. (2025) propose a systematic taxonomy for AI systems optimization, enabling benchmarking of MAS designs and evaluation of collaborative frameworks.

MAS Integration and Key Challenges In multi-agent systems, LLMs facilitate inter-agent communication, strategic planning, and iterative task decomposition (Yao et al., 2022; Wang et al., 2024a). However, scaling these agents prompts concerns about computational overhead, privacy, and the opaque "black box" nature of large models (Liu et al., 2024b; Verma, 2024). These considerations highlight the need for robust design, continuous oversight, and data-centric strategies that balance performance and interpretability.

Overall, the field has moved from manually designed agent architectures to more data-driven, automated approaches that harness LLMs' language capabilities. Despite noteworthy gains in prompt tuning, structural optimization, and integrated workflows, a gap remains for frameworks that unify these methods into efficient, adaptable, and end-to-end automated systems suited for large-scale real-world deployments.

3 Methodology

This section details the Agentic Neural Network ($\mathcal{ANN}$) methodology, a multi-agent system framework designed to solve complex, multi-step computational tasks. $\mathcal{ANN}$ is inspired by classic neural networks but replaces numerical weight optimizations with dynamic agent-based team selection and iterative textual refinement. Figure 2 shows the architecture of our $\mathcal{ANN}$.By structuring multi-agent collaboration hierarchically, $\mathcal{ANN}$ enables dynamic role assignment, adaptive aggregation, and data-driven coordination improvements through a forward-pass team selection process and a backward-pass optimization strategy. Importantly, the backward optimization is performed only during training on a task-family split to learn reusable agent-team pools and prompts, while inference on unseen tasks is strictly forward-only with a frozen architecture.

3.1 Forward Dynamic Team Selection

The $\mathcal{ANN}$ framework initiates task processing by decomposing the problem into structured subtasks. These subtasks are assigned across multiple layers, where each layer comprises a team of specialized agents working collaboratively on their designated subtask. Unlike static multi-agent workflows, $\mathcal{ANN}$ dynamically constructs these teams and their aggregation mechanisms based on task complexity. Two primary processes guide this phase: (1) defining the $\mathcal{ANN}$ structure and (2) selecting agentic teams that determine how agents collaborate.

3.1.1 Structure of the Agentic Neural Network

The architecture of $\mathcal{ANN}$ is inspired by neural networks, where each layer consists of nodes represented by agents. These agents are connected in a sequence that facilitates seamless information flow from one layer to the next, ensuring that outputs from a layer serve as structured inputs for the subsequent layer. This modular yet interconnected design enables efficient data processing, flexible task decomposition, and adaptive decision-making. Unlike static agent configurations, $\mathcal{ANN}$ dynamically refines its internal collaboration structure based on performance feedback, enhancing scalability and adaptability.

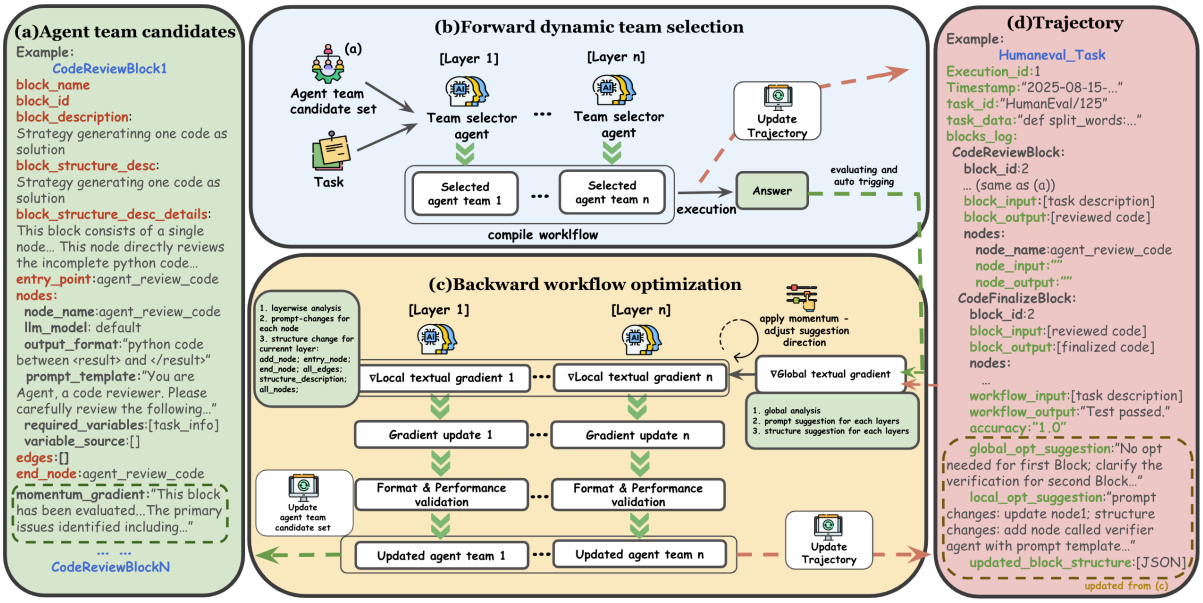

Figure 2: Overview of our $\mathcal{ANN}$ framework with textual backpropagation, which dynamically selects and refines agent teams layer by layer. (a) Agent team candidates: layer-wise candidate sets F_ℓ are represented as structured blocks composed of nodes, edges, and prompts. (b) Forward dynamic team selection: a team-selector agent selects agent team for each layer, compiles the workflow, executes the task, and records the trajectory. (c) Backward workflow optimization: task evaluation and the recorded trajectory generate a textual gradient ∇_{text}, which decomposes into a global gradient for workflow structure and local gradients for updating prompts, nodes, and edges at each layer. Momentum-based updates and format/performance validation ensure stable refinement before integrating changes into the candidate pools. (d) Trajectory: a structured record capturing the inputs and outputs of each node, layer, and workflow, along with global and local suggestions that not only guide agent team updates but are also stored as momentum gradients within the corresponding blocks, providing historical optimization references for adaptive refinement in future executions. A detailed explanation of each component is provided in Section 3

3.1.2 Selection of Layer-wise Agentic Teams

At each layer, $\mathcal{ANN}$ employs a mechanism to dynamically determine the most appropriate agentic teams, which dictates how multiple agents collaborate. This selection process considers the specific subtask requirements and complexity, ensuring that the most suitable collaborative strategy is applied to maximize performance.

Let $\mathcal{F}_\ell$ be the set of candidate agent teams available for layer ℓ, I_ℓ the input to the layer, and I the task-specific information. The dynamic team selection at each layer is determined by

$$f_\ell = \text{DynamicRoutingSelect}(\mathcal{F}_\ell, \ell, I_\ell, I),$$

where DynamicRoutingSelect denotes a process in which a team selector agent chooses the most appropriate agentic team from the candidate set based on task complexity. The selection is informed by both subtask-specific information and a structural summary of each candidate team, which are provided to the team selector agent(See Appendix B.3.Prompt 7 for details). Here, f_ℓ represents the selected agentic team. Once an agentic team is selected, the layer processes input as:

$$O_\ell = \text{ExecuteLayer}(\ell, f_\ell, I_\ell, I),$$

where the selected agentic team f_ℓ is incorporated into the overall workflow and is responsible for executing the subtask assigned to layer ℓ. The resulting output O_ℓ is then propagated as the input to the next layer, i.e., $I_{\ell+1} = O_\ell$. This dynamic selection mechanism enables $\mathcal{ANN}$ to adapt to changing task conditions, thereby optimizing efficiency and accuracy in multi-agent collaboration.

3.2 Backward Optimization

During the training phase, upon completion of the forward execution on a training task, the system evaluates its performance. If the predefined performance thresholds are not met, $\mathcal{ANN}$ triggers a backward optimization phase to refine agentic collaboration team at both the global (system-wide) and local (layer-specific) levels. Textual gradients in $\mathcal{ANN}$ should be understood as structured, language-based optimization signals and suggestions rather than formal mathematical derivatives.

3.2.1 Global Optimization

Global optimization analyzes inter-layer coordination, refining interconnections and data flow to improve overall system performance. This process provides suggestions for updating the subtask descriptions of each layer and optimizing the configuration of agent teams at the inter-layer level, primarily identifying the modules that require structural improvement. As shown in Appendix B.3.Prompt 4 for details, it further optimizes information transfer across layers to ensure that local layer behaviors align with global objectives. Mathematically, the global gradient is computed as:

$$\mathcal{G}_{\text{global}} = \text{ComputeGlobalGradient}(S, \tau),$$

where S represents the global workflow, and τ denotes the trajectory of execution, which includes agent interactions and input-output information transformations. The system structure is then updated accordingly:

$$\mathcal{S}_{\text{global}} \leftarrow \text{GlobalGradientUpdate}(\mathcal{G}_{\text{global}}, \tau).$$

The resulting optimization suggestions are subsequently used to refine the structural composition of agent teams, enabling more coherent coordination across layers.

3.2.2 Local Optimization

While global optimization refines inter-layer interactions, local optimization focuses on improving the collaboration structure of agents within each layer, adjusting their nodes, edges, and prompts based on fine-grained performance feedback. The local gradient for each layer is computed as:

$$\mathcal{G}^t_{\text{local},\ell} = \beta \mathcal{G}_{\text{global}} + (1-\beta) \times \text{ComputeLocalGradient}(\ell, f_\ell, \tau),$$

where β is a weighting factor that balances the influence of global optimization and layer-specific gradients. At the t-th step, the agentic team is updated as:

$$f^{t+1}_\ell = f^t_\ell \times \mathcal{G}^t_{\text{local},\ell},$$

where the update is guided by the local gradient feedback, which provides suggestions to refine the structural configuration and prompt design of the current agentic team. The resulting optimized team f^{t+1}_ℓ integrates these improvements to enhance layer-level performance. Appendix B.1 provides pseudo-code and Appendix B.3.Prompt 5 provides prompts used to obtain textual local gradients. **Momentum.** To improve stability, $\mathcal{ANN}$ employs momentum-based optimization, preventing sudden changes in agent parameters. The momentum-adjusted update rule is:

$$\mathcal{G}^t_{\text{local},\ell'} = \alpha \mathcal{G}^t_{\text{local},\ell} + (1-\alpha)\mathcal{G}^{t-1}_{\text{local},\ell},$$

where α is the momentum coefficient, controlling how past updates influence the current optimization step. In essence, momentum provides the model with a memory of past optimization directions. As illustrated in Appendix B.3.Prompt 6, it records how an agent team was previously adjusted (the previous adjustment direction) and combines this historical trajectory with the current feedback. By analyzing why the previous optimization failed to solve the present task, the framework refines the update direction, ensuring that new improvements build upon, rather than contradict, earlier progress. This helps the system maintain a clearer and stable sense of optimization direction across iterations.

Format Validation. It verifies that the updated structures conform to standard agent interaction formats, including proper edge definitions, variable references, and prompt templates, thereby maintaining consistency and reliability across the system.

Performance Validation. A validation agent evaluates the newly generated team against prior configurations, measuring structural similarity to prevent redundant designs and ensuring that each adjustment contributes positively to the overall functionality.

4 Experiments

4.1 Experimental Settings

Datasets We evaluate our framework on seven benchmarks: HumanEval(Chen et al., 2021), Creative Writing(Zhou et al., 2024), MATH(Hendrycks et al., 2021), DABench(Hu et al., 2024), MMLU(Hendrycks et al., 2020), Natural Plan(Zheng et al., 2024) and AIME2024&2025. HumanEval contains human-written coding problems and remains a standard benchmark for code generation. Creative Writing provides four-sentence prompts to craft a coherent story that ends with those sentences. MATH compiles challenging competition problems that demand multi-step symbolic reasoning across diverse fields. DABench covers data-analysis tasks such as feature engineering and statistics; MMLU–Machine Learning is a subset from the Massive Multitask Language Understanding (MMLU) benchmark

and offers multiple-choice questions on core ML concepts. Natural Plan focuses on scheduling tasks that require proposing feasible times under availability constraints in natural language. Following prior work (Zhou et al., 2024; Song et al., 2024; Yuksekgonul et al., 2024; Zhang et al., 2025), we split the datasets into training and validation sets for each benchmark, ensuring direct comparability with their reported baselines.

LLM Backbones To contain costs while maintaining strong performance, we unify the training process using the GPT-4o-mini model (Achiam et al., 2023) for optimization. During validation, we evaluate datasets HumanEval, CW, MATH, DABench using three backbone variants: GPT-3.5-turbo, GPT-4o-mini, and GPT-4. Because neither (Zhou et al., 2024) nor (Song et al., 2024) report GPT-4o-mini results, our findings add a new dimension to the performance landscape, showing how a budget-friendly large language model can still match or surpass top-tier methods on standard tasks. We evaluate datasets MMLU and Natural Plan using model GPT-4o and GPT-4o-0806 to follow the previous work . This setup enables us to demonstrate that our approach generalizes across different model capacities. We aim to demonstrate the flexibility and robustness of our framework in real-world various scenarios.

Baselines and Comparisons We compare $\mathcal{ANN}$ (ours) against a broad range of representative baselines, including GPTs (Brown et al., 2020; Chen et al., 2021), Agents (Zhou et al., 2023), Agents w/ AutoPE (Yang et al., 2024), DSPy/-ToT (Khattab et al., 2023b), Symbolic (Zhou et al., 2024), Vanilla LLM, Meta-prompting (Suzgun and Kalai, 2024), AutoAgents (Chen et al., 2024), DyLAN (Liu et al., 2024c), AgentVerse (Chen et al., 2023), AutoGen (Wu et al., 2023), Captain Agent (Song et al., 2024), CoT (Wei et al., 2023), TextGrad (Yuksekgonul et al., 2024), and ADAS (Hu et al., 2025). Detailed descriptions of each baseline are provided in Appendix A.2.

4.2 Experimental Results

4.2.1 Main Results

Table 1 compares our method with prior approaches on HumanEval and CW. Because (Zhou et al., 2024) provide baseline results only for GPT-3.5-turbo, hereafter referred to as GPT-3.5 and GPT-4, we supplement these with our own evaluations under GPT-4o-mini for a thorough comparison. We note the following key findings: On HumanEval, our $\mathcal{ANN}$ approach consistently surpasses all baselines. We achieve **72.7**% and **87.8**% for GPT-3.5 and GPT-4, respectively, outperforming the best baseline by a clear margin. Notably, even our GPT-4o-mini results(**90.9**/%) show competitive or superior performance despite GPT-4o-mini being a lower-cost model. For open-ended text generation tasks in CW, our method scores **9.0/7.9** on GPT-3.5/GPT-4. We attribute this to $\mathcal{ANN}$'s structured layer-wise approach, which fosters creative synergy among specialized agents while maintaining logical consistency in narrative structure.

Method	**HumanEval**	**Creative Writing**
	gpt-3.5/4o-mini/4	gpt-3.5/4o-mini/4
GPTs	59.2 / - / 71.7	4.0 / - / 6.0
Agents	59.5 / - / 85.0	4.2 / - / 6.0
Agents w/ AutoPE	63.5 / - / 82.3	4.4 / - / 6.5
DSPy / ToT	66.7 / - / 77.3	3.8 / - / 6.8
Symbolic	64.5 / - / 85.8	6.9 / - / 7.4
$\mathcal{ANN}$ **(ours)**	**72.7** / 90.9 / **87.8**	**9.0** / 8.6 / **7.9**

Table 1: Comparison results on HumanEval and Creative Writing benchmarks. The best results in each category are marked in bold.

Method	**MATH**	**DABench**
Vanilla LLM	51.53	6.61
Meta-prompting	68.88	39.69
AutoAgents	56.12	57.98
DyLAN	62.24	-
AgentVerse	69.38	-
AutoGen	74.49	82.88
Captain Agent	77.55	88.32
$\mathcal{ANN}$ (gpt-4)	<u>80.0</u>	**92.0**
$\mathcal{ANN}$ (gpt-3.5-turbo)	55.0	76.0
$\mathcal{ANN}$ (gpt-4o-mini)	**82.5**	**92.0**

Table 2: Comparison results on the MATH and DABench datasets. The best results in each column are marked in bold, and the second-best results are underlined. All results without special annotation are based on GPT-4.

In Table 2, we contrast our method with baseline results from (Song et al., 2024) on MATH and DABench. Notably, (Song et al., 2024) report results using GPT-4 but omit GPT-3.5 and GPT-4o-mini. On MATH, We record 55.0, **82.5**, and 80.0 across GPT-3.5, 4o-mini, and GPT-4. De-

Method	MMLU	Natural Plan		
	ML	Trip Planning	Meeting Planning	Calendar Scheduling
CoT	85.7	1.0	50.0	60.0
ADAS	–	3.1	43.0	66.0
TextGrad	88.4	–	–	–
$\mathcal{ANN}$	**89.1**	**7.9**	**55.0**	**73.0**

Table 3: Accuracy on MMLU-ML[1] and the Natural Plan benchmark, which consists of three task domains: TP (Trip Planning), MP (Meeting Planning), and CS (Calendar Scheduling).

spite using GPT-4o-mini in training, our method exhibits strong generalization to both GPT-3.5 and GPT-4. On GPT-4, our **80.0**% accuracy significantly outperforms Captain Agent (77.55%) and AutoGen (74.49%). On DABench, which focuses on data-analysis tasks, our method ($\mathcal{ANN}$) attains 76.0%, **92.0%**, and 92.0% on GPT-3.5, GPT-4o-mini, and GPT-4, respectively, consistently outperforming prior baselines. We observe that GPT-4o-mini again surprisingly yields top-tier results, indicating that data-centric tasks can benefit from well-structured agent orchestration without always requiring the largest language models.

Following (Zhang et al., 2025), it employs distinct models for optimization and execution. Specifically, we use GPT-4o-mini for training-phase with optimization, and select GPT-4o for validation-phase only with execution. We contrast our method with baseline results from (Yuksekgonul et al., 2024; Zhang et al., 2025) on the MMLU-ML and Natural Plan (see Table 3. For MMLU, our method achieves **89.1**% accuracy, outperforming CoT (Wei et al., 2023) (85.7%) and TextGrad (Yuksekgonul et al., 2024) (88.4%). For the Natural Plan benchmark, our method achieves accuracies of **7.9/55.0/73.0**, outperforming CoT (1.0/50.0/60.0) and ADAS (Hu et al., 2025) (3.1/43.0/66.0). This result demonstrates the advantage of our layerwise optimization approach in highly structured reasoning settings.

4.2.2 Robustness to Backbone Variation

To address concerns regarding our use of a single backbone during training, we conducted an additional experiment using GPT-3.5-turbo as the training model while retaining GPT-3.5-turbo, GPT-4o-mini, and GPT-4 as evaluation backbones. Results across HumanEval, CW, Math, and DABench benchmarks (see Table 4) show that $\mathcal{ANN}$ achieves strong generalization even when trained on GPT-3.5-turbo, a smaller-capacity model. This suggests that the agentic orchestration and textual backpropagation mechanisms in $\mathcal{ANN}$ are robust to changes in underlying language model capacity. Experiments demonstrate that the multi-agent architecture discovered by our $\mathcal{ANN}$ framework, even when using the weaker GPT-4o-mini, can generalize effectively to more powerful LLMs, achieving superior performance. Additionally, our results highlight GPT-4o-mini as a cost-effective yet high-performing alternative, reinforcing $\mathcal{ANN}$'s robustness across different model scales.

4.2.3 Ablation Studies

We conduct a unified ablation study using only GPT-4o-mini to further investigate the design choices in our $\mathcal{ANN}$ framework. Specifically, we compare four variants: (1) **Full $\mathcal{ANN}$:** Our complete approach with momentum-based optimization, validation-based performance checks, and backward optimization. (2) **w/o Momentum:** Disables the momentum technique in textual gradient refinement. (3)**w/o Validation Performance:** Skips the validation-based filtering stage when selecting improved prompts and agent roles. (4)**w/o Backward Optimization:** Does not use the backward pass to refine prompts, i.e., textual gradients for *error signals* are omitted.

Training Procedure. All four variants are trained for 20 epochs on each dataset (HumanEval, CW, MATH, DABench) using the training splits described above. To mitigate the randomness inherent in LLM sampling, we repeat each condition *three times* and report the *average* results on the validation set at regular epoch intervals.

Results and Analysis. Figure 3 illustrates the validation accuracy (or relevant score) as a function of training epoch. We observe a consistent upward trend across all four datasets, with the full $\mathcal{ANN}$ approach converging to the highest performance. Detailed findings indicate that the **impact of momentum** is substantial: removing momentum (w/o Momentum) leads to the largest performance drop on HumanEval, suggesting that gradual accumulation of textual gradient signals is crucial for code-generation tasks that require precise correctness. **Validation-based checks** also play an important role—omitting validation performance filtering can

[1]We report results on the MMLU-ML subset for fair comparison, as TextGrad provides separate results on this subset.

Train / Eval	HumanEval	Creative Writing	MATH	DABench	*Total Train Cost*
Backbones	GPT-3.5/4o-mini/4	GPT-3.5/4o-mini/4	GPT-3.5/4o-mini/4	GPT-3.5/4o-mini/4	(in USD)
GPT-3.5	73.7 / 85.5 / 86.3	8.9 / 8.5 / 8.1	53.5 / 80.0 / 77.5	71.2 / 88.0 / 91.5	≈$122.30
GPT-4o-mini	72.7 / 90.9 / 87.8	9.0 / 8.6 / 7.9	55.0 / 82.5 / 80.0	76.0 / 92.0 / 92.0	≈$73.40

Table 4: **Evaluation results across four benchmarks (HumanEval, CW, Math, and DABench)** with two different training backbones (GPT-3.5 vs GPT-4o-mini), evaluated across GPT-3.5, GPT-4o-mini, and GPT-4. Training costs are estimated based on approximately 244.6M input tokens.

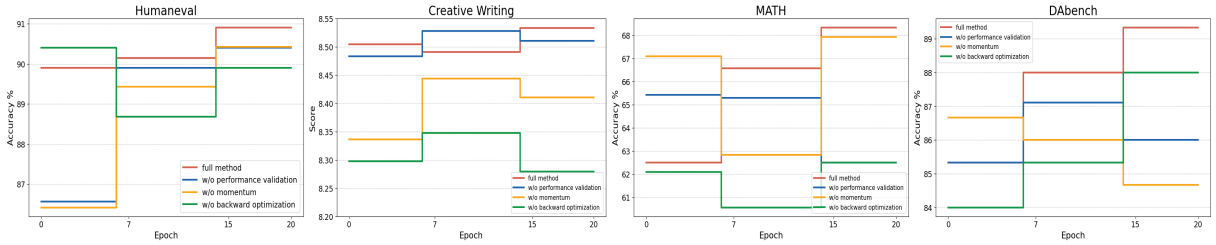

Figure 3: Ablation results on HumanEval, CW, MATH, and DABench using the GPT-4o-mini model for both training and validation. We compare the full $\mathcal{ANN}$ framework (red curve) against three ablated variants: w/o Validation Performance (blue curve), w/o Momentum (purple curve), and w/o Backward Optimization (green curve). Each curve shows average validation accuracy (or equivalent score) over three runs. The full $\mathcal{ANN}$ consistently outperforms all ablations, confirming the necessity of each component.

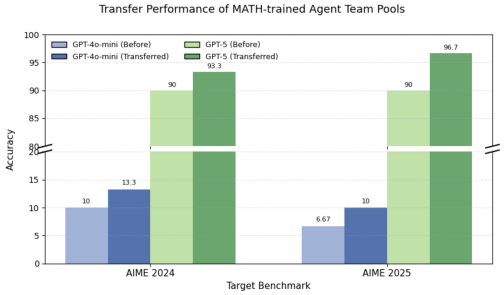

Figure 4: Cross-dataset transfer performance of MATH-trained agent team pools on AIME 2024 and AIME 2025 without further backward optimization.

cause more erratic updates, particularly evident in MATH, where narrative consistency can degrade if suboptimal agent prompts are accepted too frequently. Finally, **backward optimization** proves essential: without the backward pass, we lose a key mechanism for refining agentic team. Overall, our ablation highlights that each component contributes significantly to performance, and combining them yields the most reliable and robust improvements.

4.2.4 Cross-Dataset Transfer on Mathematical Benchmarks

To further evaluate whether $\mathcal{ANN}$ requires re-optimization for each new task or can generalize across related task families, we conduct a cross-dataset transfer study on mathematical reasoning benchmarks. We first optimize the agent team pools on the MATH training split, and then directly apply the learned candidate pools to AIME 2024 and AIME 2025 without any further backward optimization. Figure 4 shows that agent team pools optimized on MATH consistently improve performance on both AIME benchmarks after training. Notably, the transferred pools yield clear gains over the unoptimized (before) configurations under both GPT-4o-mini and GPT-5 backbones. This demonstrates that $\mathcal{ANN}$ learns reusable agentic structures that transfer across related mathematical problem distributions, rather than overfitting to a single dataset or requiring per-task re-optimization.

5 Conclusion

Our experimental results establish that $\mathcal{ANN}$ achieves high accuracy and adaptability across tasks ranging from code generation to creative writing, surpassing traditional static configurations. Through a dynamic formation of agent teams and a two-phase optimization pipeline, the framework delivers robust performance rooted in neural network design principles. These findings underscore the potential of $\mathcal{ANN}$ as an efficient solution for orchestrating complex multi-agent workflows. Detailed ablation studies highlight the significance of each component. Ultimately, this integrated agentic paradigm enables self-evolving multi-agent systems through coordinated optimization.

Limitations

Despite its advantages, the Agentic Neural Network framework has limitations. While $\mathcal{ANN}$ significantly reduces manual effort compared to prior multi-agent systems, it still requires a small amount of human initialization in the form of lightweight agent team structures and prompts. This design choice prioritizes reliability and controllability, but prevents full end-to-end automation. Future work will explore meta-prompt learning and dynamic role adjustment to further minimize human intervention while preserving stability and performance.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

Jinhe Bi, Yifan Wang, Danqi Yan, Xun Xiao, Artur Hecker, Volker Tresp, and Yunpu Ma. 2025a. Prism: Self-pruning intrinsic selection method for training-free multimodal data selection. *arXiv preprint arXiv:2502.12119*.

Jinhe Bi, Yujun Wang, Haokun Chen, Xun Xiao, Artur Hecker, Volker Tresp, and Yunpu Ma. 2024. Visual instruction tuning with 500x fewer parameters through modality linear representation-steering. *arXiv preprint arXiv:2412.12359*.

Jinhe Bi, Danqi Yan, Yifan Wang, Wenke Huang, Haokun Chen, Guancheng Wan, Mang Ye, Xun Xiao, Hinrich Schuetze, Volker Tresp, and 1 others. 2025b. Cot-kinetics: A theoretical modeling assessing lrm reasoning process. *arXiv preprint arXiv:2505.13408*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *ArXiv*, abs/2305.17126.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024. Autoagents: A framework for automatic agent generation. *Preprint*, arXiv:2309.17288.

Haokun Chen, Hang Li, Yao Zhang, Jinhe Bi, Gengyuan Zhang, Yueqi Zhang, Philip Torr, Jindong Gu, Denis Krompass, and Volker Tresp. 2025a. Fedbip: Heterogeneous one-shot federated learning with personalized latent diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pages 30440–30450.

Haokun Chen, Yueqi Zhang, Yuan Bi, Yao Zhang, Tong Liu, Jinhe Bi, Jian Lan, Jindong Gu, Claudia Grosser, Denis Krompass, and 1 others. 2025b. Does machine unlearning truly remove model knowledge? a framework for auditing unlearning in llms. *arXiv preprint arXiv:2505.23270*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Ya-Ting Lu, Yi-Hsin Hung, Cheng Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*.

Guodong Du, Zitao Fang, Jing Li, Junlin Li, Runhua Jiang, Shuyang Yu, Yifei Guo, Yangneng Chen, Sim Kuan Goh, Ho-Kin Tang, Daojing He, Honghai Liu, and Min Zhang. 2025a. Neural parameter search for slimmer fine-tuned models and better transfer. *arXiv preprint arXiv:2505.18713*.

Guodong Du, Xuanning Zhou, Junlin Li, Zhuo Li, Zesheng Shi, Wanyu Lin, Ho-Kin Tang, Xiucheng Li, Fangming Liu, Wenya Wang, Min Zhang, and Jing Li. 2025b. Knowledge grafting of large language models. *arXiv preprint arXiv:2505.18502*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, Xinbing Liang, Yaying Fei, Yuheng Cheng, and 6 others. 2024. Data interpreter: An llm agent for data science. *ArXiv*, abs/2402.18679.

Sirui Hong, Xiawu Zheng, Jonathan P. Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zi Hen Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *ArXiv*, abs/2308.00352.

Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated design of agentic systems. *Preprint*, arXiv:2408.08435.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *ArXiv*, abs/2401.05507.

Ziwei Huang, Wanggui He, Quanyu Long, Yandi Wang, Haoyuan Li, Zhelun Yu, Fangxun Shu, Long Chan, Hao Jiang, Leilei Gan, and 1 others. 2024. T2i-factualbench: Benchmarking the factuality of text-to-image models with knowledge-intensive concepts. *arXiv preprint arXiv:2412.04300*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. 2025. Mas-zero: Designing multi-agent systems with zero supervision. *Preprint*, arXiv:2505.14996.

O. Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023a. Dspy: Compiling declarative language model calls into self-improving pipelines. *ArXiv*, abs/2310.03714.

O. Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023b. Dspy: Compiling declarative language model calls into self-improving pipelines. *Preprint*, arXiv:2310.03714.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2023. Who wrote this code? watermarking for code generation. *arXiv preprint arXiv:2305.15060*.

Yu-Ang Lee, Guan-Ting Yi, Mei-Yi Liu, Jui-Chao Lu, Guan-Bo Yang, and Yun-Nung Chen. 2025. Compound ai systems optimization: A survey of methods, challenges, and future directions. *Preprint*, arXiv:2506.08234.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401.

Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. 2024a. Tapilot-crossing: Benchmarking and evolving llms towards interactive data analysis agents. *arXiv preprint arXiv:2403.05307*.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024b. Autoflow: Automated workflow generation for large language model agents. *Preprint*, arXiv:2407.12821.

Fengyuan Liu, Nouar AlDahoul, Gregory Eady, Yasir Zaki, Bedoor AlShebli, and Talal Rahwan. 2024a. Self-reflection outcome is sensitive to prompt construction. *arXiv preprint arXiv:2406.10400*.

Fengyuan Liu, Nouar AlDahoul, Gregory Eady, Yasir Zaki, Bedoor AlShebli, and Talal Rahwan. 2024b. Self-reflection outcome is sensitive to prompt construction. *arXiv preprint arXiv:2406.10400*.

Sibei Liu, Yuanzhe Zhang, Xiang Li, Yunbo Liu, Chengwei Feng, and Hao Yang. 2025. Gated multimodal graph learning for personalized recommendation. *INNO-PRESS: Journal of Emerging Applied AI*, 1(1).

Wei Liu, Haozhao Wang, Jun Wang, Ruixuan Li, Xinyang Li, Yuankai Zhang, and Yang Qiu. 2023. Mgr: Multi-generator based rationalization. *Preprint*, arXiv:2305.04492.

Wei Liu, Haozhao Wang, Jun Wang, Ruixuan Li, Chao Yue, and Yuankai Zhang. 2022. Fr: Folded rationalization with a unified encoder. *Preprint*, arXiv:2209.08285.

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

Shilin Lu, Yanzhu Liu, and Adams Wai-Kin Kong. 2023. Tf-icon: Diffusion-based training-free cross-domain image composition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2294–2305.

Shilin Lu, Zilan Wang, Leyang Li, Yanzhu Liu, and Adams Wai-Kin Kong. 2024. Mace: Mass concept erasure in diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6430–6440.

Zhengyang Lu and Ying Chen. 2019. Single image super resolution based on a modified u-net with mixed gradient loss. *Preprint*, arXiv:1911.09428.

Zhengyang Lu and Ying Chen. 2022. Pyramid frequency network with spatial attention residual refinement module for monocular depth estimation. *Journal of Electronic Imaging*, 31(02).

Zhengyang Lu and Ying Chen. 2023. Joint self-supervised depth and optical flow estimation towards dynamic objects. *Neural Processing Letters*, 55(8):10235–10249.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Cheng Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024. Scaling large-language-model-based multi-agent collaboration. *ArXiv*, abs/2406.07155.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training. *Preprint*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Xuankun Rong, Wenke Huang, Jian Liang, Jinhe Bi, Xun Xiao, Yiming Li, Bo Du, and Mang Ye. 2025. Backdoor cleaning without external guidance in mllm fine-tuning. *arXiv preprint arXiv:2505.16916*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *Preprint*, arXiv:1701.06538.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. 2024. Adaptive in-conversation team building for language model agents. *Preprint*, arXiv:2405.19425.

Mirac Suzgun and Adam Tauman Kalai. 2024. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *Preprint*, arXiv:2401.12954.

Yijun Tian, Kaiwen Dong, Chunhui Zhang, Chuxu Zhang, and Nitesh V Chawla. 2023a. Heterogeneous graph masked autoencoders. In *Proceedings of the AAAI conference on artificial intelligence*, pages 9997–10005.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. 2024. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19080–19088.

Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh Chawla. 2023b. Learning MLPs on graphs: A unified view of effectiveness, robustness, and efficiency. In *International Conference on Learning Representations*.

Ashwin Verma. 2024. *Advances in Multi-agent Decision Making Systems with Adaptive Algorithms*. Ph.D. thesis, University of California, San Diego.

Guancheng Wan, Wenke Huang, and Mang Ye. 2024. Federated graph learning under domain shift with generalizable prototypes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 15429–15437.

Chao Wang, Chuanhao Nie, and Yunbo Liu. 2025a. Evaluating supervised learning models for fraud detection: A comparative study of classical and deep architectures on imbalanced transaction data. *arXiv preprint arXiv:2505.22521*.

Fei Wang, Xingchen Wan, Ruoxi Sun, Jiefeng Chen, and Sercan Ö Arık. 2024a. Astute rag: Overcoming imperfect retrieval augmentation and knowledge conflicts for large language models. *arXiv preprint arXiv:2410.07176*.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024b. Mixture-of-agents enhances large language model capabilities. *Preprint*, arXiv:2406.04692.

Yikun Wang, Siyin Wang, Qinyuan Cheng, Zhaoye Fei, Liang Ding, Qipeng Guo, Dacheng Tao, and Xipeng Qiu. 2025b. Visuothink: Empowering lvlm reasoning with multimodal tree search. *Preprint*, arXiv:2504.09130.

Yikun Wang, Yibin Wang, Dianyi Wang, Zimian Peng, Qipeng Guo, Dacheng Tao, and Jiaqi Wang. 2025c. Geometryzero: Improving geometry solving for llm with group contrastive policy optimization. *Preprint*, arXiv:2506.07160.

Yikun Wang, Rui Zheng, Haoming Li, Qi Zhang, Tao Gui, and Fei Liu. 2024c. Rescue: Ranking llm responses with partial ordering to improve response generation. *Preprint*, arXiv:2311.09136.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Zikai Xiao, Zihan Chen, Liyinglan Liu, YANG FENG, Joey Tianyi Zhou, Jian Wu, Wanlu Liu, Howard Hao Yang, and Zuozhu Liu. 2024. Fedloge: Joint local and generic federated learning under long-tailed data. In *The Twelfth International Conference on Learning Representations*.

Zikai Xiao, Zihan Chen, Songshang Liu, Hualiang Wang, YANG FENG, Jin Hao, Joey Tianyi Zhou, Jian Wu, Howard Yang, and Zuozhu Liu. 2023. Fed-grab: Federated long-tailed learning with self-adjusting gradient balancer. In *Advances in Neural Information Processing Systems*, volume 36, pages 77745–77757. Curran Associates, Inc.

Chenghao Xu, Guangtao Lyu, Jiexi Yan, Muli Yang, and Cheng Deng. 2024. LLM knows body language, too: Translating speech voices into human gestures. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5004–5013, Bangkok, Thailand. Association for Computational Linguistics.

Rong Xuankun, Zhang Jianshu, He Kun, and Mang Ye. 2025. Can: Leveraging clients as navigators for generative replay in federated continual learning. In *ICML*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *ArXiv*, abs/2309.03409.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

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

Xinlei Yu, Ahmed Elazab, Ruiquan Ge, Jichao Zhu, Lingyan Zhang, Gangyong Jia, Qing Wu, Xiang Wan, Lihua Li, and Changmiao Wang. 2025. Ich-prnet: a cross-modal intracerebral haemorrhage prognostic prediction method using joint-attention interaction mechanism. *Neural Networks*, 184:107096.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *Preprint*, arXiv:2406.07496.

Zhi Zeng, Minnan Luo, Xiangzheng Kong, Huan Liu, Hao Guo, Hao Yang, Zihan Ma, and Xiang Zhao. 2024. Mitigating world biases: A multimodal multi-view debiasing framework for fake news video detection. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 6492–6500.

Gengyuan Zhang, Jinhe Bi, Jindong Gu, Yanyu Chen, and Volker Tresp. 2023. Spot! revisiting video-language models for event understanding. *arXiv preprint arXiv:2311.12919*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2024. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Yao Zhang, Chenyang Lin, Shijie Tang, Haokun Chen, Shijie Zhou, Yunpu Ma, and Volker Tresp. 2025. Swarmagentic: Towards fully automated agentic system generation via swarm intelligence. *Preprint*, arXiv:2506.15672.

Jinman Zhao and Xueyan Zhang. 2024. Large language model is not a (multilingual) compositional relation reasoner. In *First Conference on Language Modeling*.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2024. Natural plan: Benchmarking llms on natural language planning. *Preprint*, arXiv:2406.04520.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023. Agents: An open-source framework for autonomous language agents. *Preprint*, arXiv:2309.07870.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. 2024. Symbolic learning enables self-evolving agents. *Preprint*, arXiv:2406.18532.

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

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024b. Language agents as optimizable graphs. *ArXiv*, abs/2402.16823.

Contents

Framework	Layerwise	Backward Optimization	Momentum Adjustment	Global Optimization	Local Optimization	Dynamic Teaming	Training Requirement
Symbolic (Zhou et al., 2024))	✗	✓	✗	✓	✓	✗	✓
AutoGen (Wu et al., 2023)	✗	✗	✗	✓	✓	✓	✗
InfiAgent-DAbench (Hu et al., 2024)	✗	✗	✗	✓	✗	✓	✗
MetaGPT (Hong et al., 2023)	✗	✗	✗	✗	✓	✓	✗
DyLan (Liu et al., 2024c)	✗	✓	✗	✓	✓	✓	✓
Adaptive Team (Song et al., 2024)	✗	✗	✓	✗	✓	✓	✗
Chain-of-Thought (Wei et al., 2023)	✗	✗	✗	✗	✓	✗	✗
GPTSwarm (Zhuge et al., 2024b)	✗	✗	✓	✓	✓	✓	✓
Aflow (Li et al., 2024b)	✗	✗	✗	✓	✗	✓	✗
$\mathcal{ANN}$ **(Ours)**	✓	✓	✓	✓	✓	✓	✓

Table 5: Framework-level comparison across layerwise design, optimization strategies (backward, momentum, global/local), dynamic team composition, and training requirements. ✓/✗ indicate support.

A Comparison and baseline

A.1 Comparision

A.1.1 Framework-Level Comparison

With the rapid advancement and widespread adoption of deep learning techniques (Liu et al., 2022, 2023; Lu and Chen, 2019, 2022, 2023; Tian et al., 2023a, 2024, 2023b; Wan et al., 2024; Liu et al., 2025; Xiao et al., 2024, 2023), large language models (Bi et al., 2024, 2025b,a; Du et al., 2025b,a; Wang et al., 2025a) have emerged as a transformative force across diverse domains (Chen et al., 2025a; Rong et al., 2025; Zhang et al., 2023; Chen et al., 2025b; Zhao and Zhang, 2024; Yu et al., 2025; Huang et al., 2024; Zeng et al., 2024; Xu et al., 2024; Lu et al., 2024, 2023; Xuankun et al., 2025; Liu et al., 2022; Wang et al., 2025c, 2024c, 2025b). Their ability to understand, generate, and reason over natural language has enabled a new generation of intelligent systems, particularly in the orchestration and coordination of multi-agent frameworks. As these models continue to evolve, numerous architectures have been proposed to harness their capabilities in increasingly sophisticated and dynamic environments.

To situate $\mathcal{ANN}$ in the rapidly evolving ecosystem of multi-agent orchestration, we benchmark it against nine representative frameworks drawn from recent literature—*Symbolic* (Zhou et al., 2024), *AutoGen* (Wu et al., 2023), *InfiAgent-DAbench* (Hu et al., 2024), *MetaGPT* (Hong et al., 2023), *DyLan* (Liu et al., 2024c), *Adaptive Team* (Song et al., 2024), *Chain-of-Thought* (Wei et al., 2023), *GPTSwarm* (Zhuge et al., 2024b), and *Aflow* (Li et al., 2024b). Collectively, these baselines cover symbolic planning, agentic workflow coordination, dynamic team formation, and optimisation-driven routines, thus furnishing a balanced backdrop for assessing architectural and functional advances.

Table 5 distils the comparison along seven orthogonal dimensions: (i) *layerwise decomposition*, (ii) *back-propagated optimisation*, (iii) *momentum-based adjustment*, (iv) *global optimisation scope*, (v) *local-only optimisation*, (vi) *dynamic team selection*, and (vii) *task-specific training requirements*. A check mark (✓) indicates native support; a cross (✗) denotes absence. As the table shows, $\mathcal{ANN}$ is the only framework that provides *full* coverage across all criteria—combining layerwise granularity with momentum-augmented backward optimisation, unifying global and local objectives, and eliminating the need for costly task-specific fine-tuning through on-the-fly team selection.

A.1.2 Trainable Architectures versus Test-Time Reasoning

Our framework is inspired by planning- and reflection-style reasoning methods such as Chain-of-Thought (CoT) (Wei et al., 2023), Tree-of-Thought (ToT) (Khattab et al., 2023b), Graph-of-Thought (GoT) (Besta et al., 2024), and Reflexion, which demonstrate that explicitly structuring intermediate reasoning steps,

search trees, or reflection loops can significantly improve large language model performance on complex tasks. However, $\mathcal{ANN}$ differs fundamentally from these methods in formulation, optimization, and reuse of multi-agent workflows.

Test-time reasoning vs. training-time optimization. Planning and reflection methods operate purely at *test time*. For each new input instance, they dynamically expand reasoning chains, trees, or reflection trajectories and discard them after inference. They do not maintain a persistent architecture, nor do they perform training-phase optimization over agent structures. As a result, performance is scaled primarily by increasing test-time computation rather than by learning reusable workflow structures.

In contrast, $\mathcal{ANN}$ treats a multi-agent workflow as a *trainable, layered architecture*. The framework defines a fixed number of layers, each maintaining a candidate pool of structured agent teams (blocks composed of nodes, edges, and prompts). These candidate pools are optimized during a lightweight training phase using a small training split, and the resulting architecture is reused for all future instances at test time without further search or expansion. This decouples architectural learning from test-time inference and positions $\mathcal{ANN}$ as a data-efficient training procedure rather than a test-time scaling heuristic.

Local reasoning updates vs. workflow-level optimization. Reflection-based methods typically revise a single reasoning trace, prompt, or trajectory in isolation. In contrast, optimization in $\mathcal{ANN}$ is performed through a forward–backward loop over the *entire layered workflow*. During the backward phase, textual back-propagation decomposes feedback into a global component, which identifies structurally problematic layers and blocks, and local components, which refine nodes, edges, and prompts within selected blocks. These textual gradients are accumulated with momentum, providing a notion of optimization trajectory across iterations and enabling stable architectural refinement.

Implicit reasoning structures vs. explicit agentic architectures. In $\mathcal{ANN}$, agent team structures are treated as discrete parameters subject to optimization. At inference time, a team-selector agent performs dynamic routing by selecting the most suitable block from each layer's candidate pool based on the current subtask and execution trajectory. Importantly, the backward phase does not blindly accept LLM-generated updates: every proposed block must pass format, structure, and performance validation before entering the candidate pool. This yields a neural-network-like training loop that searches over a discrete space of multi-agent architectures, as summarized in Table 5.

Taken together, while $\mathcal{ANN}$ draws conceptual inspiration from planning and reflection methods, it differs fundamentally in scope and mechanism. Rather than expanding reasoning at test time, our framework learns a reusable, layered multi-agent architecture through training-phase optimization, enabling stable generalization across tasks and instances.

A.2 Baseline Details

We compare $\mathcal{ANN}$ (ours) with the following baseline approaches:

GPTs (Brown et al., 2020; Chen et al., 2021): A direct usage of GPT-based models with carefully designed prompts.

Agents (Zhou et al., 2023): A language-agent method that organizes multi-step reasoning and tool usage through a pipeline of prompts.

Agents w/ AutoPE (Yang et al., 2024): A variant wherein each prompt node is optimized by an LLM, but without full language gradient backpropagation.

DSPy / ToT (Khattab et al., 2023b): A pipeline optimization framework that performs search-based tuning of prompt components, applicable mostly to tasks with a tractable evaluation function.

Symbolic (Zhou et al., 2024): An agent-based system employing symbolic learning methods for dynamic prompt improvements.

Vanilla LLM: A single-turn GPT-based approach without agent collaboration.

Meta-prompting (Suzgun and Kalai, 2024): An adaptive prompting strategy that attempts to generate meta-level instructions for new tasks.

AutoAgents (Chen et al., 2024): An automated agent system that attempts to orchestrate multi-agent interactions but can be unstable in large-scale settings.
DyLAN (Liu et al., 2024c): A dynamic language-agent approach that decomposes tasks with feedback loops.
AgentVerse (Chen et al., 2023): A multi-agent platform emphasizing flexible agent composition.
AutoGen (Wu et al., 2023): A system featuring an "Assistant + Executor" design for multi-step problem-solving.
Captain Agent (Song et al., 2024): An adaptive team-building agent framework that spawns specialized sub-agents based on task progress.
CoT (Chain-of-Thought) (Wei et al., 2023): A prompting strategy that encourages intermediate reasoning steps, often used to enhance zero-shot performance on complex QA tasks.
TextGrad (Yuksekgonul et al., 2024): A framework that performs solution-level optimization using textual gradients.
ADAS (Automated Design of Agentic Systems) (Hu et al., 2025): A framework in which a meta-agent autonomously generates agentic architectures for multi-agent system design.

B Implementation

B.1 Pseudo Code

This section provides pseudocode for the system's overall architecture and the local gradient optimization process. Algorithm 1 outlines how the network leverages a dynamic routing mechanism alongside an agentic neural network structure, integrating both global optimization and layerwise optimization. Dynamic routing selects the most suitable path for a given task, thereby enhancing overall system performance and stability. Global optimization steers the entire network toward optimal solutions, while layerwise optimization fine-tunes each layer for improved learning efficiency and reliability. Algorithm 2 focuses on local optimization within each specialized layer. By applying localized gradient updates, each module can concentrate on its respective sub-task. Such targeted adjustments accelerate convergence, improve learning efficiency, and, in conjunction with the global optimization strategy, enhance the system's overall performance.

B.2 Initialization and Deployment

Although $\mathcal{ANN}$ supports dynamic structural evolution through textual backpropagation, the amount of manual initialization required in our implementation is intentionally kept minimal. Our design follows the same practical assumption adopted by prior multi-agent frameworks, namely that a small set of initial building blocks is provided, while the majority of structural refinement and prompt evolution is handled automatically.

Specifically, existing systems such as AutoGen (Wu et al., 2023), GPTSwarm (Zhuge et al., 2024b), and ADAS (Hu et al., 2025) also rely on manually specified templates or primitives as their starting point. AutoGen requires developers to instantiate agents with predefined system prompts and fixed conversational patterns. GPTSwarm initializes a hand-designed operation graph whose nodes and edges are later optimized. ADAS searches over a code space composed of author-defined agent templates and controller patterns. In comparison, $\mathcal{ANN}$ requires only a lightweight initialization that is comparable to, or smaller than, these prior approaches.

In practice, deploying $\mathcal{ANN}$ for a new task family only requires providing a compact JSON configuration file that defines a small number of initial candidate agent teams (typically 1–3 blocks per layer) and a few reusable agent candidates. After this initialization step, all subsequent structural evolution, including adding or removing nodes, modifying edges, and refining prompts, is performed automatically by the LLM during the backward optimization phase. Users are not required to manually design or maintain prompts for all future agents, which keeps the deployment cost low.

Config 1 shows a simplified initialization example used for the HumanEval code-review layer. Despite its simplicity, the same initialization format is reused across diverse domains, including code generation

Algorithm 1: Agentic Neural Network with Dynamic Routing and Adaptive Optimization

Require: I: dataset input; L: layers in the workflow; F_ℓ: set of possible aggregation functions for each layer ℓ; S: workflow updation for optimization

Ensure: Updated structure and prompts for the agentic neural network

Traj $\leftarrow$ [] ▷ Initialize Trajectory
$I_\ell \leftarrow I$ ▷ Initialize input of first layer
Forward Pass with Dynamic Routing and Aggregation
for each layer ℓ in L **do**
    $f_\ell \leftarrow \text{DynamicRoutingSelect}(F_\ell, \ell, I_\ell, I)$
    $O_\ell \leftarrow \text{ExecuteLayer}(\ell, f_\ell, I_\ell, I)$
    Append $(\ell, f_\ell, I_\ell, O_\ell)$ to Traj
    $I_{\ell+1} \leftarrow O_\ell$
end for
Backpropagation:
Global Optimization
$\mathcal{G}_{\text{global}} \leftarrow \text{ComputeGlobalGradient}(S, \text{Traj})$
$\mathcal{S}_{\text{global}} \leftarrow \text{GlobalGradientUpdate}(\mathcal{G}_{\text{global}}, \text{Traj})$
Layerwise Optimization
for each layer ℓ in reverse(L) **do**
    $\mathcal{G}^t_{local,\ell} \leftarrow \text{ComputeLocalGradient}(\ell, f_\ell, \text{Traj}, \mathcal{L}_{\text{global}})$
    if momentum_needed **then**
        $\mathcal{S}_{\text{local}} \leftarrow \text{LocalGradientUpdate}(\ell, f_\ell, \mathcal{G}^t_{local,\ell}, \mathcal{S}_{\text{global}})$
    else
        $\mathcal{G}^t_{local,\ell'} \leftarrow \text{ApplyMomentum}(\ell, \text{Traj}, \mathcal{G}^t_{local,\ell}, \mathcal{G}^{t-1}_{local,\ell})$
        $\mathcal{S}_{\text{local}} \leftarrow \text{LocalGradientUpdate}(\ell, f_\ell, \mathcal{G}^t_{local,\ell'}, \mathcal{S}_{\text{global}})$
    end if
end for
return (F_ℓ, Traj)

(HumanEval), data analysis (DABench), mathematical reasoning (MATH), creative writing, knowledge-intensive question answering (MMLU–ML), and natural-language planning (Natural Plan), without redesigning the workflow from scratch. In our experiments, the full initialization file for each dataset consists of only a few hundred lines of JSON and requires only minor domain-specific adjustments.

Table 6 summarizes the scale of manual initialization used across all benchmarks. For all six datasets, the number of manually specified agent-team candidates is at most three, demonstrating that $\mathcal{AN}\mathcal{N}$ achieves broad applicability with very limited human intervention.

Config 1: Example JSON Initialization for the HumanEval Code-Review Layer

```
{
  "blocks": {
    "CodeReviewBlock1": {
      "block_name": "CodeReviewBlock1",
      "block_id": "1",
      "block_description": "Please carefully review the following incomplete
           Python code snippet, understand its structure, logic, and existing
           functionality....",
      "block_structure_description": "Strategy selecting better solution
           between two generated codes.",
      "block_structure_description_details": "...",
      "entry_point": "agent_pseudo_code",
      "nodes": {
```

Algorithm 2: LocalGradientUpdate

Require: ℓ: current layer; f_ℓ: selected aggregation function; Traj: trajectory of execution; $\mathcal{G}_{\text{global}}$: global gradient; $\mathcal{S}_{\text{global}}$: current global structure; F_ℓ: set of possible aggregation functions for each layer ℓ

Ensure: Updated global structure $\mathcal{S}_{\text{global}}$ and valid aggregation function f_ℓ

$\mathcal{G}_{local,\ell} \leftarrow \text{ComputeLocalGradient}(\ell, f_\ell, \text{Traj}, \mathcal{G}_{\text{global}})$ ▷ Compute local gradient in layer ℓ
$\mathcal{S}_{\text{local}} \leftarrow \text{LocalGradientUpdate}(\ell, f_\ell, \mathcal{G}_{local,\ell}, \mathcal{S}_{\text{global}})$: ▷ $\mathcal{S}_{\text{local}}$: Update layer-wise workflow
for $k \leftarrow 1$ to 3 **do** ▷ Attempt up to 3 updates
    $f'_\ell \leftarrow \text{LocalGradientUpdate}(\ell, f_\ell, \mathcal{G}_{local,\ell}, \mathcal{S}_{\text{global}})$
    ValidateUpdate (f'_ℓ): ▷ If update passes validation
    Node Validation:
    if VariableSourcesValid(f'_ℓ) & FormatValid(f'_ℓ) **then**
        Edge Validation:
        if AllNodesHaveEdges(f'_ℓ) **then**
            Structure Validation:
            if StructureNotUnique(f'_ℓ) **then**
                if ValidatePerformance(f'_ℓ, f_ℓ) **then**
                    Append f'_ℓ to F_ℓ ▷ add new agg func f'_ℓ into F_ℓ
                    break ▷ Exit update loop on success
                end if
            end if
        end if
    end if
end for
return $\mathcal{S}_{\text{global}}$

Benchmark	# Initial Agent Teams	# Initial Agents
HumanEval	2	8
Creative Writing	2	7
MATH	3	7
DABench	3	7
MMLU–ML	2	8
Natural Plan	2	8

Table 6: Scale of manual initialization across benchmarks.

```
"agent_pseudo_code": {
  "agent": "agent_pseudo_code",
  "llm_model": "default_llm",
  "prompt": "..."
},
"agent1_review_code": {
  "agent": "agent1_review_code",
  "llm_model": "default_llm",
  "prompt": "..."
},
"agent2_review_code": {
  "agent": "agent2_review_code",
  "llm_model": "default_llm",
  "prompt": "..."
},
"agent3_decision_maker": {
  "agent": "agent3_decision_maker",
  "llm_model": "default_llm",
  "prompt": "..."
```

```
        }
      },
      "edges": [
        ["agent_pseudo_code", "agent1_review_code"],
        ["agent_pseudo_code", "agent2_review_code"],
        ["agent1_review_code", "agent3_decision_maker"],
        ["agent2_review_code", "agent3_decision_maker"]
      ],
      "end_node": "agent3_decision_maker"
    },
    "CodeReviewBlock2": {
      "block_name": "CodeReviewBlock2",
      "block_id": "2",
      "block_description": "Please carefully review the following incomplete
                  Python code snippet, understand its structure, logic, and
                  existing functionality....",
      "block_structure_description": "Strategy generating one code as solution.",
      "block_structure_description_details": "...",
      "entry_point": "agent_review_code",
      "nodes": {
        "agent_review_code": {
          "agent": "agent_review_code",
          "prompt": "..."
        }
      },
      "edges": [],
      "end_node": "agent_review_code"
    }
  },
  "nodes": {
    "agent_review_code_after_pseudo_code": {
      "agent": "agent_review_code",
      "prompt": "..."
    },
    "agent_decision_maker_with_2_options": {
      "agent": "agent_decision_maker_with_2_options",
      "prompt": "..."
    },
    "agent_static_analysis": {
      "agent": "agent_static_analysis",
      "prompt": "..."
    }
  }
}
```

B.3 Prompt Repository

To guarantee rigorous experimentation, our framework distills complex evaluation and optimisation routines into a curated suite of six reusable examples of *prompts* for reference. Each prompt encapsulates a distinct facet of model assessment—ranging from factual exactness to strategic, multi-layer workflow repair—thereby furnishing a unified interface for loss-function design and optimiser selection. Collectively, these templates enable (i) **fine-grained answer verification**, (ii) **holistic workflow diagnosis**, and (iii) **progressive, momentum-aware refinement**, furnishing the gradient signals that steer the training loop towards globally coherent behaviour.

Answer Verification. Prompt 2 formalises a strict comparison between a model's predicted answer and an externally supplied ground truth, while Prompt 3 generalises the rubric to creative-writing settings where no canonical answer exists.

Global Optimisation. Prompt 4 performs gradient-based analysis over an entire workflow trajectory, isolating error-prone sub-tasks and prescribing block-level remedies.

Layer-wise Repair. Prompt 5 zooms in on a single block, recommending structural or prompt-template adjustments that preserve inter-block consistency.

Momentum-based Adjustment. Prompt 6 fuses historical "velocity" information with fresh gradient signals to resolve recurrent faults while safeguarding previously effective changes.

Block Selection. Prompt 7 scores competing blocks against task complexity, ensuring that the most capable module is invoked for code-finalisation tasks and analogous challenges.

By systematically orchestrating these prompts, we induce *task-aligned* gradients that couple local correctness with global workflow efficiency, thereby enhancing both convergence speed and final performance.

Prompt 1: Prompt for Answer Verification with Ground Truth

```
You are a helpful AI assistant.
You will use your math skills to verify the answer.

You are given:
- A problem: {problem}
- A reply from a model: {final_answer}
- A ground truth answer: {solution}

Please do the following::
1. Extract the answer from the reply in the format:
   "The answer is <answer extracted>"

2. Compare the extracted answer with the ground truth.

3. Based on your analysis, choose only one of the following outputs:
   (a) "The answer is correct."
   (b) "The answer is approximated but should be correct."
   (c) "The answer is incorrect.
       Correct Answer: <ground truth answer> </ground truth answer> |
       Answer extracted: <answer extracted> </answer extracted>."
   (d) "The reply doesn't contain an answer."
```

Prompt 2: Prompt for Creative Writing Evaluation

```
Evaluate the following creative writing piece based on the provided task.

Inputs:
- Task Description: {task_prompt}
- Creative Writing Output: {output_from_last_layer}

Evaluation Criteria:
- Logical coherence: Is the text logically organized?
- Emotional engagement: Does the text evoke the desired emotions?
- Adherence to task requirements: Does it match the original prompt?
- Creativity: Is the text original and imaginative?

Output Format:
- Coherence: [Score out of 10, with a brief explanation]
- Engagement: [Score out of 10, with a brief explanation]
- Adherence: [Score out of 10, with a brief explanation]
- Creativity: [Score out of 10, with a brief explanation]
- Suggestions for Improvement: [Text]
- Overall Score: [Score out of 10]
```

Prompt 3: Prompt for Gradient-Based Global Optimization

```
Task:
You are an advanced global workflow analysis assistant tasked with diagnosing
inefficiencies and proposing optimizations for a multi-step process.
Your goal is to analyze the workflow trajectory and determine which aspects
```

```
need improvement to address task failures and enhance overall performance.

You will evaluate the provided consolidated information from a workflow task.
Identify which sub-task outputs or prompts likely caused the failure and
provide specific suggestions for each sub-task. Your output must strictly
follow this format:
<output\_format>\{example\_global\_loss\_format\}</output\_format>

Important Notice:
- All analyses and suggestions should be based on a general level.
- Avoid overly targeted feedback for this specific task instance.
- All required information is provided via: {initial_solution}

Global Optimization Steps:

1. Final Result Evaluation:
   Analyze the final result <final result> to determine if the task failed.

2. Solution Comparison:
   Compare <canonical solution> and <generated solution>:
   - Is the logic in <generated solution> aligned with <canonical solution>?
   - Where is the gap between the analysis and the standard answer?
   - Identify specific issues in <generated solution> that contributed to the
     failure.
   - Document these findings in the 'global_analysis' section of the
     <output_format>.

3. Block Input and Output Analysis:
   Based on the <task description> and <workflow trajectory>:
   - Do not compare the block outputs with the <canonical solution>.
   - Examine each block_input and block_output.
   - Identify which block(s) caused the task to fail.
   - Highlight any inefficiencies or redundancies.
   - Write optimization suggestions into the 'structure_suggestion' section
     of each relevant block.
   - Review each block's block_description and provide edits if necessary,
     recorded in the 'prompt_suggestions' section.
   - If no edits are needed, do not add any suggestions.

4. Node-Level Analysis Within Blocks:
   For each problematic block:
   - Analyze the internal node_input and node_output.
   - Evaluate the team collaboration structure.
   - Propose improvements to intra-block agent collaboration, if necessary.
   - Document your suggestions in the 'structure_suggestion' section of the
     corresponding block.
```

Prompt 4: Prompt for Layer-Wise Block Optimization

```
You are given a block within a workflow. Your task is to suggest
optimizations for this block, focusing on both prompt improvements and
structural changes, while ensuring consistency and efficiency.

Block Information:
- Block Name: {block_name}
- Global Loss Feedback: {global_loss_feedback}
  (This is global feedback for the entire workflow. Use as reference, but
  base suggestions on block-level reasoning.)
- Blocks Log: {blocks_log}
  (Includes architecture, node inputs/outputs, block/node descriptions.)
- Canonical Solution: {canonical_solution}
- Task Description: {task_prompt}

Evaluation Criteria:
1. Evaluate Each Node
- Check input_variables for validity and consistency.
```

```
- Valid sources include:
  * State variables: "task_data", "task_prompt", "task_id"
  * Prior node outputs: e.g., calculation_expert1_output
- For prompt modifications:
  * Include an updated prompt_template with clear instructions
  * Explicitly list all input_variables and their sources

2. Propose Structural Changes
- Add/remove nodes (max 3 additions)
- For added nodes, specify:
  * node_name, agent, output format, prompt_template
  * variable_sources, constraints
- Define from/to edges for new nodes
- Update connected nodes' input_variables if needed
- Set the new entry_node and end_node
- Ensure all nodes (except end_node) have valid outgoing edges
- Include all_edges_now and all_nodes_now

3. Impact on Other Nodes
- Maintain logical consistency with the entire workflow

4. Use Available Agents
- Refer to {available_agents} for potential agents
- Check each agent's constraints for fit
- Modify agents as needed (update prompt_template, input_variables, or define
new agents)

5. Dynamic Block ID and Naming
- Use {new_block_id} to assign a unique block_id
- Format name as {block_name}X, where X = new_block_id

6. Block Structure Description
- Include:
  * block_structure_description: high-level purpose
  * block_structure_description_details: including:
    1. Nodes and connections
    2. Node roles and logic
    3. Input/output flow
- Ensure clarity, accuracy, and alignment with structure

7. Provided Canonical Solution and Test Cases
- Don't over-optimize: block may not be the cause of failure
- Avoid overfitting: feedback should remain generalized
- Use <canonical solution> and <test cases> as reference only

8. Output Format
- All feedback must be returned in this JSON format: {layerwise_loss_format}
- Do not use arrows to represent edges!
```

Prompt 5: Prompt for Momentum-Based Adjustment

```
Task Description:
You are an advanced strategic advisor focused on enhancing team performance.
Your role is to analyze recent feedback in combination with historical
adjustments to guide team improvement for a specific workflow block.

Provided Information:
- Team Name: <team name> {block_name} </team name>
- Current Team Structure: <current team> {current_block} </current team>
- Final Result of Task Execution:
  <final result> {current_task_results} </final result>
- Current Gradient Feedback:
  <current feedback> {current_gradient} </current feedback>
- Previous Adjustment Direction:
  <previous adjustment direction> {velocity}</previous adjustment direction>
- Input and Output for Block and Nodes:
```

```
  * <team input> {block_input} </team input>
  * <team output> {block_output} </team output>
  * <input and output of all nodes> {nodes_info}
    </input and output of all nodes>

Instructions - Two-Step Strategy:

1. Overlap Handling:
- If <current feedback> overlaps with </previous adjustment direction>, focus
on these overlapping issues.
- Since the current version <current team> was formed via previous
adjustments, but <final result> still failed, analyze why earlier suggestions
did not work.
- Carefully review block_input, block_output, and nodes_info to pinpoint
reasons for failure.
- Revise the <current feedback> so it addresses overlapping issues in a more
effective way.

2. New Issues Maintenance:
- If <current feedback> introduces new problems not found in
  <previous adjustment direction>, retain those.
- Slightly refine and consolidate all suggestions to form an updated version
  of feedback.

Important Notes:
- This block may not be the root cause of task failure.
  Avoid over-optimization.
- Our optimization is dataset-level, not task-specific. Do not overfit
  feedback to this task instance.

Output Format:
Return your suggestions using the same structure as <current feedback>,
wrapped as:
<adjusted feedback> [Your updated suggestions here] </adjusted feedback>
```

Prompt 6: Prompt Example for Layer Selection Based on Task Difficulty

```
Task Description:
You are a performance evaluator tasked with selecting the most suitable block
for solving a Python code finalization task.
The complete workflow consists of three blocks: code_review_block,
code_finalize_block, and code_execute_block.

Current Block:
The block under evaluation is code_finalize_block, which represents the second
layer in the workflow.
It's purpose is to refine another agent's code output based on prior messages,
considering:
- Syntax accuracy
- Logical completeness
- Adherence to the initial coding intent

If the code meets the above standards, keep it unchanged. Otherwise, provide a
corrected version.

Task Details:
- Task Objective: Improve the agent's output code using the contextual
messages.
- Task Description: <task description> {task_prompt} </task description>

Available Blocks:
Below is a list of available blocks, including their structural roles and
descriptions:
<list of all block's structure description> {blocks_structure_descriptions}
</list of all block's structure description>
```

```
Instructions:
1. Evaluate the <task description> carefully, identifying key difficulty
points and requirements.
2. Compare block roles and structures from <list of all block's structure
description> to determine which best fits the task.
3. Select the most appropriate block based on the task complexity.

Output Format:
- Output your selection using the exact format below:
  <selected_agg_func> X </selected_agg_func>
- For example, selecting CodeFinalizeBlock3 should result in:
  <selected_agg_func> 3 </selected_agg_func>
```

C Experiment Details

C.1 Training and Validation Split

We adopt an 80/20 protocol per dataset when comparable to prior work, and otherwise follow the dataset-specific setups cited below. This keeps splits aligned with baselines while ensuring evaluation on unseen tasks. The table fuses all details (counts, protocol, and per-task runtime).

Benchmark	Total	Train	Val	Split	Train / task	Test / task
HumanEval[a]	164	131	33	80/20	7 s ↑ 30 s	7 s
Creative Writing[a]	100	80	20	80/20	17 s ↑ 30 s	17 s
MATH (subset)[b]	196	157	39	80/20	13 s ↑ 33 s	13 s
DAbench[c]	257	206	51	80/20	15 s ↑ 34 s	15 s
MMLU–ML[d]	128	16	112	custom	8 s ↑ 28 s	8 s

Table 7: Dataset splits and approximate per-task runtimes. ↑ indicates additional time from backward optimization during training; inference is forward-only. Notes: (a) 8:2 as in Zhou et al. (2024); (b) 196-problem subset and sampling as in Song et al. (2024); (c) full dataset with 8:2; (d) ML subset as in TEXTGRAD (Yuksekgonul et al., 2024), base dataset Hendrycks et al. (2020).

C.2 Effect of Training Data Size

To further quantify the generalization behavior of $\mathcal{ANN}$ and evaluate its sensitivity to the amount of available training data, we conduct an additional analysis under different training regimes. All experiments in this subsection follow the dataset-level optimization protocol described in Appendix D.4 and are evaluated exclusively on held-out test tasks that are never used during textual backpropagation.

Specifically, we consider three training settings for each benchmark: **(i) No Train**, where $\mathcal{ANN}$ executes the initial forward-only workflow without any backward optimization; **(ii) 50% Train**, where textual backpropagation is performed using only half of the available training tasks; and **(iii) Full Train**, where the framework is optimized using the complete training split. In all cases, evaluation is conducted on the same unseen test set. All results reported in this subsection are obtained using GPT-4o-mini to ensure consistency.

Table 8 summarizes the results across HumanEval, Creative Writing, MATH, and DABench. We report the total number of tasks, the sizes of the training and test splits, and the corresponding test performance under each training regime. In addition, the table reports the growth of candidate agent team pools across layers, illustrating how textual backpropagation enriches the set of available agent teams during training.

Overall, the results show that textual backpropagation consistently improves test-time performance over the forward-only baseline, even when applied to a reduced training set. Training with 50% of the available tasks already yields noticeable gains over the untrained configuration, while full training further enhances performance. These trends indicate that $\mathcal{ANN}$ learns reusable agentic structures that generalize beyond the specific training instances, rather than relying on per-task re-optimization.

Benchmark	Total	Train	Test	Results (No Train)	Results (50% Train)	Results (Full Train)
HumanEval	164	131	33	86.8	87.9	90.9
Creative Writing	100	80	20	8.3	8.5	9.0
MATH	196	157	39	65.0	70.0	82.5
DABench	257	206	51	86.3	86.3	92.0

Table 8: Test performance under different training regimes. "No Train" denotes forward-only execution without textual backpropagation; "50% Train" denotes optimization using half of the available training tasks; "Full Train" denotes optimization using the complete training split. All results are evaluated on unseen test tasks and use GPT-4o-mini.

D Case Study

D.1 Prompt Evolutions Examples

Figure 5 and Figure 6 illustrate representative trajectories of prompt evolution across two benchmark tasks: subtask about code review in the HumanEval dataset and subtask about task analysis in the DABench suite, respectively. These diagrams reflect both the structural transformations of block-level workflows and the fine-grained progression of node-level prompt design. Together, these visualizations exemplify how the prompt design co-evolved with structural modularity.

HumanEval: Code Review Prompt Evolution. Figure 5 demonstrates how the system's prompt architecture evolved in the context of solving the `review_code` subtask on the HumanEval dataset. Initially, the workflow consisted of a single-agent node responsible for completing partially written code. As the system matured, this simplistic design was incrementally augmented with a multi-agent framework involving two parallel reviewers and a subsequent decision node. Each reviewer agent received increasingly structured prompts, incorporating pseudo-code context, explicit reasoning criteria (e.g., correctness, efficiency, readability), and modular output constraints.

In subsequent iterations, the system integrated static analysis agents, forming a pluggable review-correction pipeline. The final prompt configuration emphasized modular roles, strict output formatting, and conditional rewriting policies, resulting in a robust, interpretable code review pipeline.

DABench: Task Analysis Prompt Evolution. Figure 6 illustrates the evolution of task analysis prompts when solving data-centric reasoning problems in the DABench benchmark. The initial system was anchored around a single agent generating a natural-language strategy and accompanying pseudo-code. Prompt instructions were general-purpose, with minimal context sensitivity or structural annotation.

With successive iterations, the system adopted a multi-agent architecture, introducing review, feedback, and revision loops. Each agent's prompt was incrementally specialized: reviewers were instructed to analyze structural logic, adherence to constraints, and planning completeness. Prompts began incorporating input-specific metadata, including task constraints, file paths, and structured output tags (e.g., `<analysis>`, `<feedback>`, `<result>`).

D.2 Team Structure Examples with Optimization

To better understand how agent team structures evolve throughout the optimization process, we present visualizations of team configurations across multiple datasets. These examples demonstrate how architectures transition from simple, linear pipelines to more dynamic, graph-based systems as the model learns to coordinate more effectively.

Figure 7 illustrates selected examples from three representative datasets: Creative Writing (Zhou et al., 2024), Math (Hendrycks et al., 2021), and MMLU–Machine Learning (Hendrycks et al., 2020). For each dataset, we choose a single layer and show how the team structure at that layer evolves over time. As optimization progresses, the agent configurations become increasingly complex and tailored to the demands of each dataset, reflecting greater specialization and improved collaboration.

Figure 8 focuses on two additional datasets: HumanEval (Chen et al., 2021) and DABench (Hu et al., 2024). In the case of DABench, we adopt the random train/validation split from (Song et al., 2024). Here,

we emphasize the functional diversity among agents by using different node colors to indicate distinct roles (e.g., generation, evaluation, decision-making). These visualizations highlight how functional heterogeneity and task-specific routing emerge through optimization.

Together, these figures demonstrate how adaptive reconfiguration of agent teams enables more effective problem solving and reflects the system's ability to internalize dataset-specific strategies.

D.3 Stability and Failure Modes

Text-based optimization with large language models can introduce noise, hallucination, or drift, especially when structural updates are generated in free-form natural language. While the main body of this paper focuses on successful optimization trajectories, it is equally important to understand when and how textual backpropagation fails, and how such failures are prevented from propagating across layers. In $\mathcal{ANN}$, stability is explicitly enforced through a combination of momentum-based updates and a multi-stage validation mechanism that filters out unstable, ill-formed, or non-improving updates before they are integrated into the candidate pool.

Momentum-based stabilization. Beyond validation, $\mathcal{ANN}$ employs momentum to stabilize structural optimization across iterations. Rather than treating each textual gradient independently, momentum maintains a directional memory of previous update steps, encoding how and why an agent team was modified in earlier iterations. This historical trajectory is integrated with newly computed textual gradients, which helps suppress oscillatory updates and enables error-aware correction when previous optimizations fail. As a result, structural changes evolve smoothly over time instead of reacting abruptly to noisy feedback.

Multi-stage validation and error interception. As summarized in Algorithm 2, every locally updated agentic workflow must pass a sequence of validation checks before it is accepted. These include *node validation* (e.g., VARIABLESOURCESVALID, FORMATVALID), *edge validation* (e.g., ALLNODESHAVEEDGES), *structure validation* (e.g., STRUCTURENOTUNIQUE), and *performance validation*. Only candidate blocks that satisfy all validation criteria are allowed to enter the candidate pool and influence subsequent layers.

In practice, a non-trivial fraction of LLM-generated updates are rejected by these checks. This rejection mechanism plays a crucial role in preventing hallucinated structures, invalid variable dependencies, or redundant architectures from propagating errors through the layered workflow. Table 9 presents representative failure cases observed during training, illustrating how different validation stages intercept problematic updates. For example, node validation can reject updates that introduce invalid variable sources, structure validation can prevent duplicate block configurations, and performance validation can filter out changes that do not yield measurable improvements.

Taken together, these mechanisms ensure that textual backpropagation in $\mathcal{ANN}$ is not a blind acceptance of LLM suggestions. Instead, optimization is tightly constrained by explicit validation rules, performance-based filtering, and momentum-based smoothing, ensuring that errors remain localized and do not cascade across layers. Consistent with this design, our ablation studies show that removing momentum or validation components leads to noticeably less stable optimization behavior.

D.4 Dataset-Level Optimization and Generalization

A key design principle of $\mathcal{ANN}$ is the explicit separation between *offline dataset-level optimization* and *online test-time execution*. Rather than re-optimizing the system for each individual task instance, our goal is to learn reusable agent teams for a *task family* (e.g., HumanEval coding problems, MATH competition questions, or DABench data-analysis tasks) and then apply the learned architecture to unseen tasks from the same distribution.

Concretely, for each benchmark, $\mathcal{ANN}$ is trained once using a designated training split. During this phase, textual backpropagation is applied only to training tasks: execution trajectories from multiple tasks are aggregated, and both global and local textual gradients are used to update candidate agent teams, prompts, and inter-layer connections. No backward optimization is performed on validation or test

Validation type	Task ID	Block (candidate)	Failure reason	Log summary
Node Validation	HumanEval /67	CodeReviewBlock3	Invalid variable source assignment: node `agent_input_validation` sets `variable_sources["input_string"]` to an unregistered source (`node_input`).	Invalid variable source detected; update rejected.
Structure Validation	HumanEval /127	CodeFinalizeBlock6	Duplicate structure detected: edge set `all_edges_now` matches an existing block (`CodeFinalizeBlock4`).	Duplicate block structure; update rejected.
Performance Validation	HumanEval /145	CodeFinalizeBlock8	No performance gain: `validate_performance` returns `False` despite passing structural checks.	Predicted performance not improved; update rejected.

Table 9: Representative failure cases intercepted by different validation stages during textual backpropagation.

examples. After training, the resulting candidate team pools and workflow structure are frozen and reused for all subsequent test-time evaluations.

At inference time, $\mathcal{ANN}$ operates in a purely forward manner. Each layer maintains a fixed candidate pool of agent teams, and a team-selector agent dynamically routes the input by selecting the most suitable team based on the current subtask and execution context. Importantly, no further textual gradients, architectural updates, or prompt modifications are performed during evaluation. As a result, test-time performance reflects the generalization capability of the learned agentic architecture rather than per-instance adaptation.

This protocol is consistently applied across all benchmarks. For HumanEval, Creative Writing, MATH, and DABench, we adopt an 80%/20% train–test split, while for MMLU–ML we follow the dataset-specific split used in prior work. All results reported in the main paper are obtained by optimizing $\mathcal{ANN}$ on the training split and evaluating on disjoint, unseen test tasks. Quantitative evidence under different training regimes, including forward-only execution and partial training, is reported in Appendix C.2.

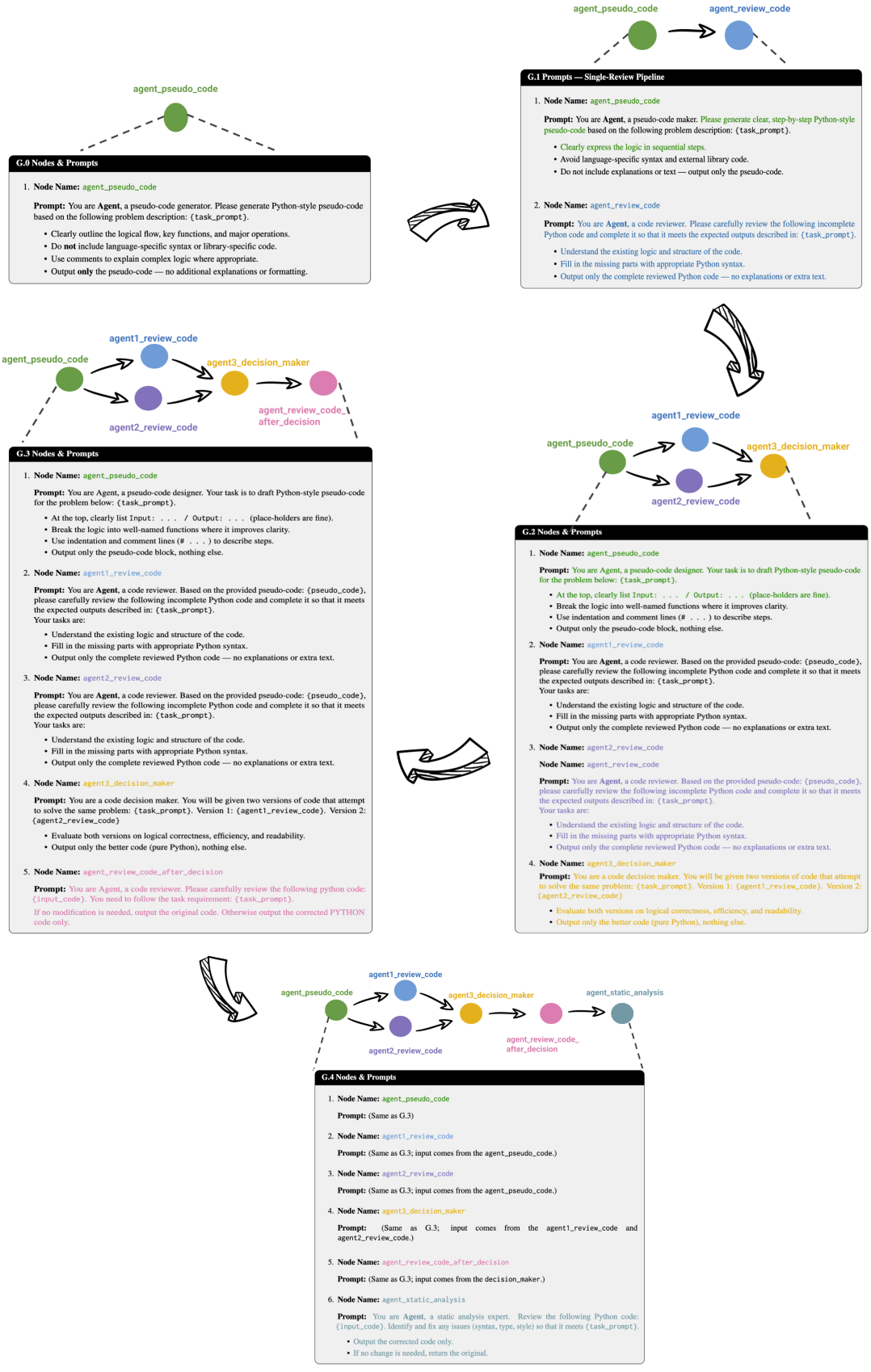

Figure 5: Prompt-evolution trajectory for the HumanEval(Chen et al., 2021) *review_code* subtask. Boxes denote agent nodes, arrows indicate information flow, and shaded regions highlight components newly introduced at each iteration.

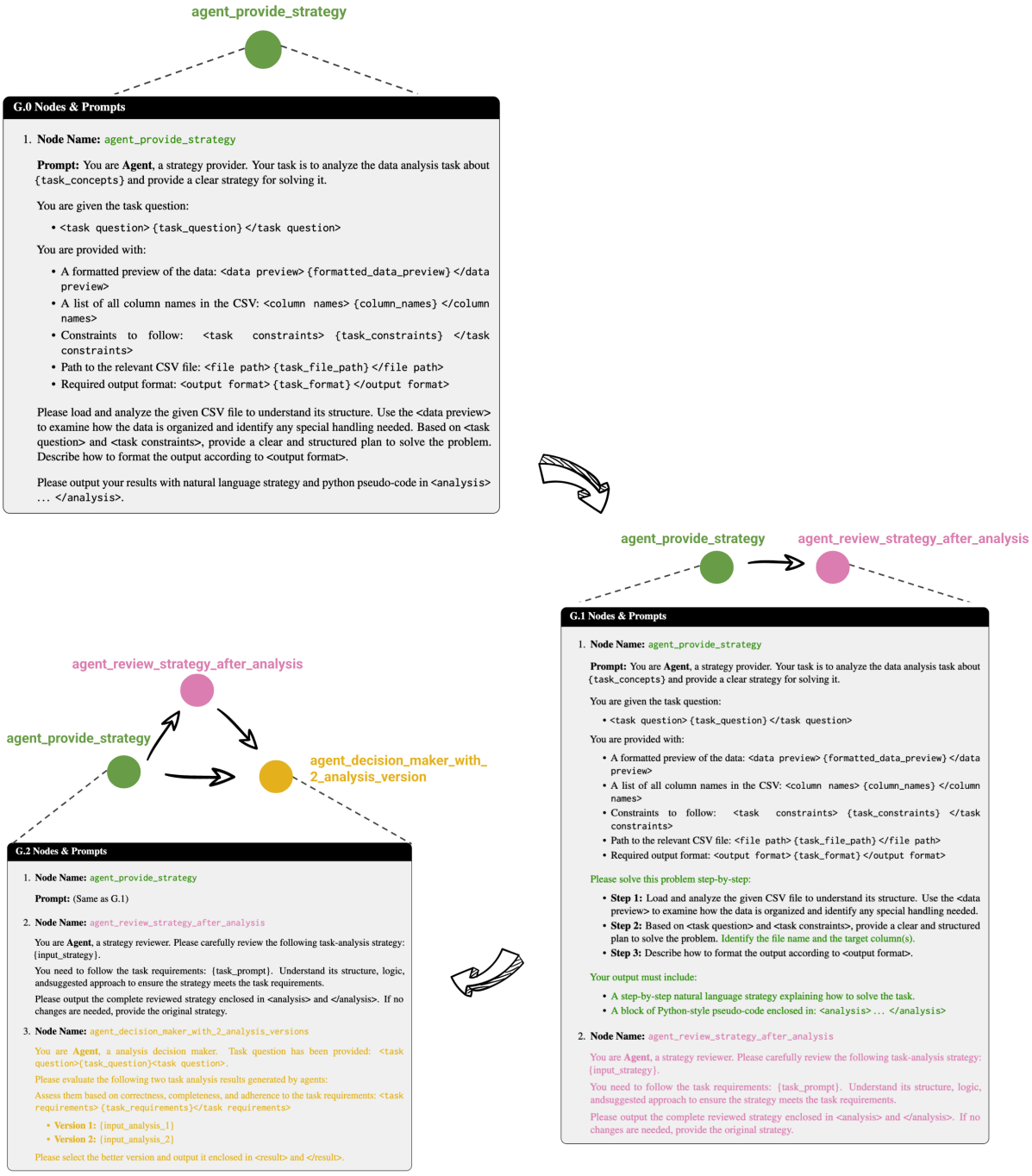

Figure 6: Prompt-evolution trajectory for the DABench(Hu et al., 2024) task-analysis benchmark. Boxes denote agent nodes, arrows indicate information flow, and shaded regions highlight components newly introduced at each iteration.

Figure 7: Evolution of agent team structures on the **Creative Writing** (Zhou et al., 2024), **Math** (Hendrycks et al., 2021), and **MMLU–Machine Learning** (Hendrycks et al., 2020) datasets. For each dataset, we visualize a representative example from one layer, showing how team configurations become progressively more structured and cooperative through optimization.

Figure 8: Team structure visualizations for the **HumanEval** (Chen et al., 2021) and **DABench** (Hu et al., 2024) datasets. Each node's color reflects its functional role within the system. The diagrams highlight how different types of agents coordinate and how task-specific configurations emerge over time.