Agentic Neural Networks: A Neuro-Symbolic Approach to Multi-Agent Systems with 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 (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 adaptively improve their roles, prompts, and coordination. This neuro-symbolic approach enables ANN to create new or specialized agent teams post-training, delivering notable gains in accuracy and adaptability. Across four benchmark datasets, ANN surpasses leading multi-agent baselines under the same configurations, showing consistent performance improvements. Our findings indicate that ANN provides a scalable, data-driven framework for multi-agent systems, combining the collaborative capabilities of LLMs with the efficiency and flexibility of neural network principles. We plan to open-source the entire framework.

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., 2023; 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., 2024) 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., 2024), 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; M. Hu and Zhou (2024) 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. These innovations mirror earlier developments in neural network 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.; Zhou et al., 2024). By integrating agents, prompts, and tools into a single optimization process, these frameworks pave the way for data-centric approaches in which performance and learning signals, rather than manual design, guide architectural decisions (Hinton and Salakhutdinov, 2006; Yao et al., 2022).

Building on these insights, we introduce the *Agentic Neural Network (ANN)*, a framework that

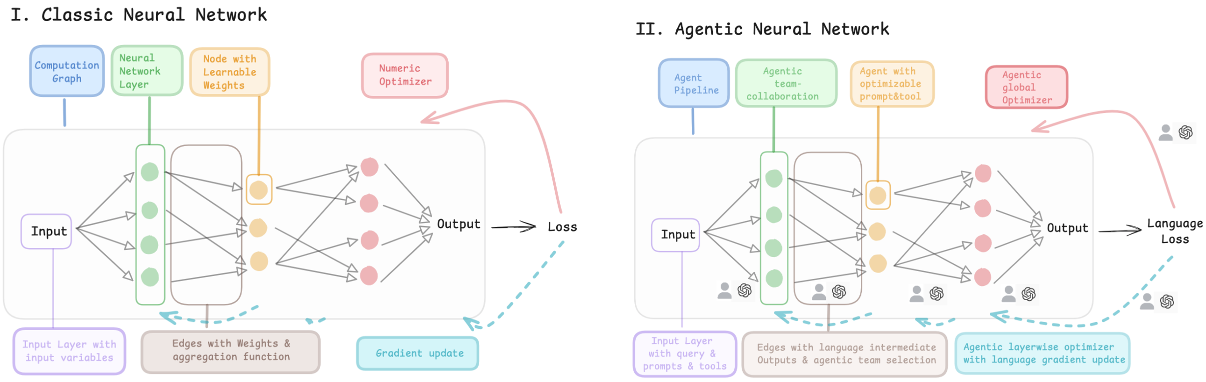

Figure 1: A conceptual comparison between classic neural networks (left) and our ANN (right). In the classic paradigm, learnable weights and numeric optimizers enable end-to-end training via gradient-based updates. In ANN, each layer corresponds to a team of language agents whose roles, prompts, and tools can be jointly optimized through textual gradients.

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 ANN considers each layer as a team of language agents, jointly optimizing roles, prompts, and tools through textual gradients (Yuksekgonul et al., 2024). Instead of a purely engineering-driven approach, 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, 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 this framework's capabilities, we evaluate ANN on four challenging tasks. First, **MATH** probes advanced mathematical reasoning, requiring agents to manage multi-step proofs and symbolic manipulations. Second, **DABench** centers on data science tasks such as filtering, transformation, and analysis. Third, **Creative Writing** demands coherent narrative construction and consistent text generation. Lastly, **HumanEval** evaluates the system's coding abilities, with strict demands on correctness and efficiency. Our experiments show that ANN not only simplifies MAS design by automating prompt tuning, role assignment, and agents collaboration but also outperforms existing baselines in accuracy.

By uniting symbolic agent coordination with connectionist optimization, *ANN* provides a cohesive, data-driven solution that lowers reliance on manual and heuristic engineering. Our results indicate that a fully unified perspective—one in which LLM-based agents, prompts, and workflows are co-optimized—could pave the way for more robust and flexible multi-agent systems.

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 and Narasimhan, 2018; Radford et al., 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., 2024).

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., 2024). *Hyperparameter tuning* fine-tunes model parameters or scheduling (Liu et al., 2024a), and *workflow optimization* revises entire computational graphs or code structures (M. Hu and Zhou, 2024; Zhang et al., 2024; Zhuge et al.). Symbolic learning frameworks (Hong et al., 2024; Zhuge et al., 2024; Zhou et al., 2024) optimize prompts, tools, and node configurations collectively, mitigating local optima that can emerge from optimizing each component independently.

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., 2024). 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 (ANN) methodology, a multi-agent system framework designed to solve complex, multi-step computational tasks. Figure 2 shows the comparison between static and dynamic approaches. ANN is inspired by classic neural networks but replaces numerical weight optimizations with dynamic agent-based team selection and iterative textual refinement. By structuring multi-agent collaboration hierarchically, 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.

3.1 Forward Dynamic Team Selection

The 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, ANN dynamically constructs these teams and their aggregation mechanisms based on task complexity. Two primary processes guide this phase: (1) defining the ANN structure and (2) selecting aggregation functions that control how agent outputs are combined.

3.1.1 Structure of the Agentic Neural Network

The architecture of 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, ANN dynamically refines its internal collaboration structure based on performance feedback, enhancing scalability and adaptability.

3.1.2 Selection of Layer-wise Aggregation Functions

At each layer, ANN employs a mechanism to dynamically determine the most appropriate aggregation function, which dictates how outputs from

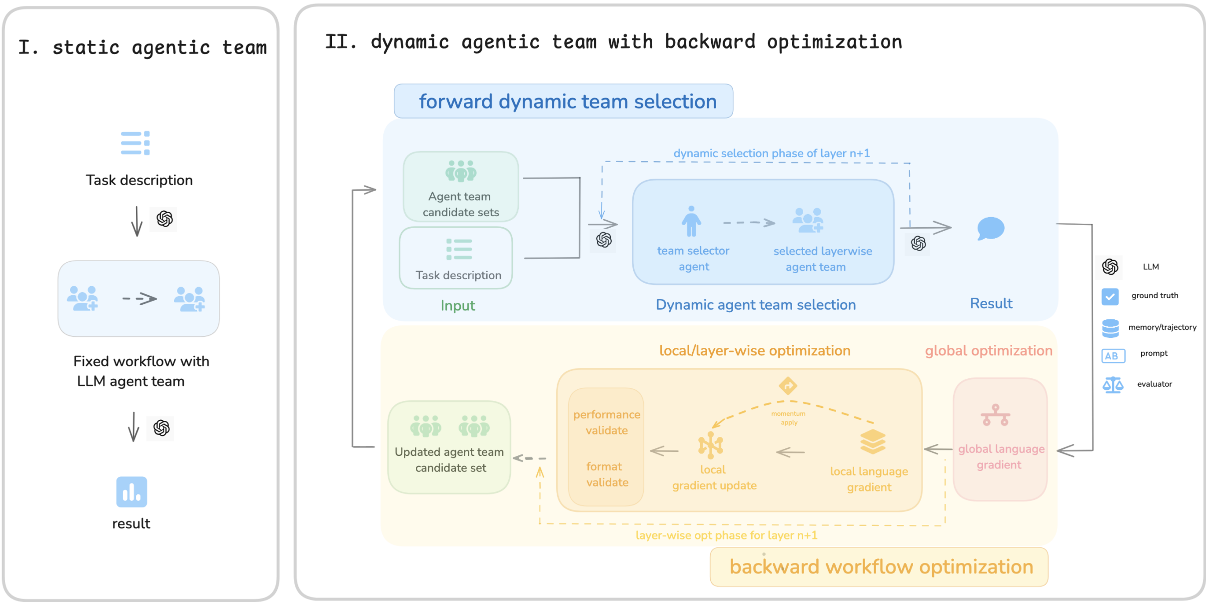

Figure 2: Difference between static agentic team and our framework. The left panel illustrates a static agentic team, where a fixed workflow is predefined for a given task without adaptability. In contrast, the right panel demonstrates our ANN framework, which dynamically selects and refines agent teams layer by layer. During the forward phase, ANN constructs task-specific agent teams through dynamic selection mechanisms. If performance does not meet predefined criteria, the backward phase triggers layer-wise local optimizations and global refinements through textual feedback and gradient updates.

multiple agents are combined. 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 aggregation functions available for layer ℓ, I_ℓ the input to the layer, and I the task-specific information. The aggregation function selection at each layer is determined by

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

where DynamicRoutingSelect selects candidate functions based on task complexity and prior execution trajectory and f_ℓ represents the selected aggregation function. Once an aggregation function is selected, the layer processes input as:

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

where O_ℓ serves as the input to the next layer with $I_{\ell+1} = O_\ell$. This dynamic aggregation mechanism ensures that ANN adapts to changing task conditions, optimizing efficiency and accuracy in multi-agent collaboration.

3.2 Backward Optimization

Upon completion of the forward phase, the system evaluates its performance. If the predefined performance thresholds are not met, ANN triggers a backward optimization phase to refine agent interactions and aggregation functions at both the global (system-wide) and local (layer-specific) levels.

3.2.1 Global Optimization

Global optimization analyzes inter-layer coordination, refining interconnections and data flow to improve overall system performance. This process adjusts aggregation functions and optimizes information transfer across layers to better 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

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

3.2.2 Local Optimization

While global optimization refines inter-layer interactions, local optimization fine-tunes agents and aggregation functions within each layer, adjusting their parameters based on detailed performance feedback. This targeted approach addresses inefficiencies and bottlenecks identified during execution, enhancing overall adaptability. 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. In t-th step, the aggregation function is updated as

$$f_\ell^{t+1} = f_\ell^t - \eta \mathcal{G}^t_{\text{local},\ell},$$

where η is a step size parameter that regulates updates.

Several additional techniques are incorporated throughout the pipeline. Figure 2 compares the full framework with a static workflow. Additionally, the appendix provides pseudo-algorithms and prompts used to obtain textual global feedback and local gradients.

Momentum To improve stability, 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.

Format Validation Ensures that all agent interactions comply with predefined communication protocols, maintaining system reliability and coherence.

Performance Validation Regular performance assessments validate the efficacy of the optimizations, ensuring that each adjustment contributes positively to the system's overall functionality.

4 Experiments

In this section, we provide a comprehensive overview of our experimental setup, datasets, baselines, and results. We evaluate the proposed Agentic Neural Network (ANN) across four datasets: **HumanEval**, **Creative Writing**, **MATH**, and **DABench**. These datasets are chosen for their diversity and prior usage in related work, allowing us to situate our contributions within established benchmarks. We divide our experiments into two main categories: *(i)* HumanEval and Creative Writing, following the protocols described in (Zhou et al., 2024), and *(ii)* MATH and DABench, aligning with the evaluation approaches in (Song et al., 2024).

4.1 Datasets

HumanEval. The HumanEval dataset (Chen et al., 2021) consists of human-written coding problems requiring the model to generate executable code that correctly solves each problem. It has long been used to benchmark code-generation performance for language models.

Creative Writing. Following (Zhou et al., 2024), the Creative Writing dataset is comprised of short textual prompts (each consisting of four random sentences) and requires the model or agent to compose a coherent narrative ending in these predetermined sentences. Unlike standard benchmark tasks, Creative Writing emphasizes open-ended generation, coherence, and creativity.

MATH. We also evaluate on MATH (Hendrycks et al., 2021), a collection of high-level competition math problems encompassing diverse mathematical fields. This dataset is widely recognized as a rigorous benchmark for logical reasoning and symbolic manipulation. We note that MATH problems often involve step-by-step reasoning and multi-stage computations, providing a challenging testing ground for multi-agent coordination and textual gradient refinement.

DABench. Finally, we use DABench (Hu et al., 2024) for data-analysis tasks, including feature engineering, statistical computations, and real-world data manipulations. As per (Song et al., 2024), we employ a random split into training and validation sets. DABench's tasks not only require robust coding and data manipulation skills but also demand a coherent workflow for reading, transforming, and interpreting data.

4.2 Experimental Settings

Overview of Training and Validation. Following the practice in both (Zhou et al., 2024) and (Song et al., 2024), we split dataset into training set and validation set for each dataset. However, each reference employs a slightly different splitting strategy:

1. HumanEval & Creative Writing. We adopt the ratio and split procedure described in (Zhou

et al., 2024), ensuring direct comparability to their reported baselines.

2. MATH & DABench (Adaptive protocol). We follow (Song et al., 2024), who suggest a random subset for training and another subset for validation in their ablation studies. Each dataset's split ratio is consistent with their recommended setting.

LLM Backbones To contain costs and yet maintain strong performance, we unify the training process using only the GPT-4o mini model (Achiam et al., 2023). Concretely, all fine-tuning, agent configuration, and prompt optimizations are carried out on 4o mini. Then, during validation, we evaluate each dataset with three backbone variants: **GPT-3.5**, GPT-4o mini, and GPT-4. This procedure allows us to:

1. Demonstrate how our approach generalizes across different model capacities,

2. Compare against prior work that primarily reports results on GPT-3.5 or GPT-4,

3. Highlight that *4o mini*, even though it is lower-cost, achieves competitive (often superior) performance relative to existing baselines, effectively bridging a cost-effectiveness gap in agent-based experimentation.

Because neither (Zhou et al., 2024) nor (Song et al., 2024) report 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. By training on *GPT-4o mini* (see details below) and validating on multiple LLM backbones, we aim to demonstrate the flexibility and robustness of our framework in real-world various scenarios.

Baselines and Comparisons. We compare **ANN** (ours) with various baseline approaches, each drawn from the references: **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 back-propagation. **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 to break down 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.

Method	HumanEval	Creative Writing
	3.5/4o mini/4	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
ANN (ours)	**72.7** / 93.9 / **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.

Unless otherwise stated, the baseline results in Table 1 (HumanEval and Creative Writing) are taken from (Zhou et al., 2024), while those in Table 2 (MATH and DABench) are from (Song et al., 2024). Since none of these works tested on 4o mini, we omit highlighting the best results for 4o mini in the tables.

4.3 Experimental Results

4.3.1 Main Results

Table 1 compares our method with prior approaches on HumanEval and Creative Writing. Because (Zhou et al., 2024) provide baseline results only for GPT-3.5 and GPT-4, we supplement these with our own evaluations under 4o mini for a thorough comparison. We note the following key findings:

Method	MATH	DABench
	3.5 / 4o mini / 4	3.5 / 4o mini / 4
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
ANN (ours)	55.0 / 82.5 / **80.0**	76.0 / 95.0 / **92.0**

Table 2: Comparison results on the MATH and DABench datasets. The best results in each category are marked in bold.

- **Humaneval:** Our ANN approach consistently surpasses all baselines. On HumanEval, 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 4o mini results **93.9/**% show competitive or superior performance despite 4o mini being a lower-cost model.

- **Creative Writing:** For open-ended text generation, our method scores **9.0/7.9** on GPT-3.5/GPT-4. We attribute this to ANN's structured "layerwise" approach, which fosters creative synergy among specialized agents while maintaining logical consistency in narrative structure.

Next, in Table 2, we contrast our method with baseline results from (Song et al., 2024) on MATH and DABench. Again, (Song et al., 2024) report GPT-3.5 and GPT-4 but omit 4o mini. Observations include:

- **MATH:** We record 55.0, 82.5, and **80.0** across GPT-3.5, 4o mini, and GPT-4. Despite using 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%).

- **DABench:** On data-analysis tasks, ANN attains 75.6, **95.0**, and 88.88 on GPT-3.5, 4o mini, and GPT-4, respectively, consistently outperforming prior baselines. We observe that 4o mini again surprisingly yields top-tier results (95.0), indicating that data-centric tasks can benefit from well-structured agent orchestration without always requiring the largest language models.

Experiments demonstrate that the multi-agent architecture discovered by our 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 ANN's robustness across different model scales.

4.4 Ablation Studies

We conduct a unified ablation study using only 4o mini to further investigate the design choices in our ANN framework. Specifically, we compare four variants:

1. **Full 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., omits textual gradients for "error signals."

Training Procedure. All four variants are trained for 20 epochs on each dataset (HumanEval, Creative Writing, 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 ANN approach converging to the highest performance. Detailed findings:

- **Impact of Momentum:** Removing momentum (w/o Momentum) leads to the largest performance drop on HumanEval, suggesting that

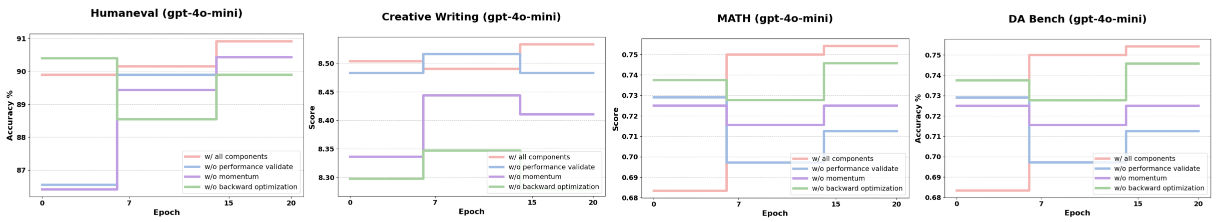

Figure 3: Ablation results on HumanEval, Creative Writing, MATH and DABench, each using 4o mini for training and validation. We compare the full ANN framework against variants that remove momentum (w/o Momentum), remove validation performance checks (w/o Validation), and remove backward textual optimization (w/o Backward). Performance trends upward with more epochs, showing the critical role of each component.

gradual accumulation of textual gradient signals is crucial for code-generation tasks that require precise correctness.

- **Validation-Based Checks:** Omitting validation performance filtering can cause more erratic updates, particularly evident in MATH, where narrative consistency can degrade if suboptimal agent prompts are accepted too frequently.

- **Backward Optimization:** Without the backward pass, we lose a key mechanism for pinpointing errors and refining agent roles. This shortfall manifests in weaker improvements per epoch, especially on the mathematically oriented Creative Writing dataset.

Overall, our ablation highlights that each component contributes significantly to performance, and combining them yields the most reliable and robust improvements.

5 Future Work

Although our current ANN framework provides a flexible mechanism for agent configuration and task partitioning, it still depends substantially on manually defined initial structure candidates and node prompts, limiting its adaptability to diverse domains. A more automated strategy, such as meta-prompt learning (S. Hu and Clune, 2024; Yin et al., 2024), could reduce reliance on human-crafted templates by generating initial layouts from accumulated agent experience. Another challenge is that as the number of candidate teams grows, computational overhead increases, making it less efficient to identify the most effective teams. Advanced pruning techniques, such as periodic pruning and performance-driven filtering, could be integrated in future work to enhance efficiency while preserving diversity. Moreover, current agent roles are largely static once a team is instantiated, restricting flexibility for highly intricate or evolving tasks. Introducing a dynamic role adjustment mechanism that reacts in real time to changing requirements would enhance adaptability and task performance. Finally, although momentum-based optimization and structured optimization strategies have been proposed, they have not yet been deeply integrated into one cohesive approach. Addressing these directions—meta-prompt learning, pruning, dynamic role reassignment, and enhanced optimization—would equip ANN to become a more powerful, efficient, and versatile platform for dynamic multi-agent collaboration.

6 Conclusion

Our experimental results establish that 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 ANN as a scalable and efficient solution for orchestrating complex multi-agent workflows. Detailed ablation studies highlight the significance of each component. Ultimately, this integrated agentic paradigm paves the way for fully automated multi-agent systems, effectively combining symbolic coordination with connectionist optimization.

7 Limitations

Despite its advantages, the Agentic Neural Network framework has limitations. Its reliance on manually defined structures and prompts reduces adaptability across tasks, which could be improved through meta-prompt learning to automate structure generation. Moreover, candidate selection becomes computationally expensive as the pool grows, requiring periodic pruning, though this risks homogenization, which could be mitigated by stochastic retention of lower-ranked candidates. Furthermore, while ANN dynamically selects aggregation functions, agent roles remain fixed, limiting adaptability to evolving tasks, which could be improved by allowing agents to adjust roles based on real-time feedback. Future work will address these limitations by integrating meta-prompt learning, adaptive pruning, and dynamic role adjustments to enhance ANN's scalability and adaptability.

Potential Risks While the Agentic Neural Network (ANN) offers enhanced adaptability and efficiency in multi-agent collaboration, it also introduces potential risks. One concern is the interpretability of emergent agent behaviors, as dynamically evolving agent teams may develop complex interaction patterns that are difficult to analyze or debug. Additionally, increased computational demands from iterative optimization cycles could limit scalability in resource-constrained environments, requiring careful management to balance performance with efficiency.

References

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

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, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 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.

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

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, 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.

Geoffrey E. Hinton and Ruslan Salakhutdinov. 2006. Supporting online material for reducing the dimensionality of data with neural networks.

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, Zhibin Gou, Zongze Xu, Chenglin Wu, Li Zhang, Min Yang, and Xiawu Zheng. 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.

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.

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. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

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. 2023. Large language models are zero-shot reasoners.

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*.

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. 2024. Tapilot-crossing: Benchmarking and evolving llms towards interactive data analysis agents. *arXiv preprint arXiv:2403.05307*.

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

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

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024c. A dynamic llm-powered agent network for task-oriented agent collaboration.

T. Lin M. Hu and C. Zhou. 2024. Adas: An automatic design framework for agent systems using llm-based optimization. *arXiv preprint arXiv:2401.05507*.

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, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

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 and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

C. Lu S. Hu and J. Clune. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

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.

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

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

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.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation.

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.

Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and William Yang Wang. 2024. G\" odel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*.

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.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2024. Aflow: Automating agentic workflow generation.

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. 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. 2024. Language agents as optimizable graphs. *ArXiv*, abs/2402.16823.

A appendix

A.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.

A.2 Prompt Examples

This section serves as a Prompt Template dedicated to defining the Loss Function and Optimizer used in our system. By systematically tailoring the loss function and choosing the most suitable optimizer strategies, this template enhances the model's ability to learn effectively and improve overall performance.

A.3 Team Structure Examples with optimization

This section describes the evolutionary process of nodes within the system, illustrating how they transition from an initial linear architecture to more sophisticated, graph-based structures. By monitoring performance, synergy, and task requirements, the network dynamically reconfigures its connections. This adaptive strategy allows for enhanced connectivity, efficient information flow, and robust cooperative behavior among nodes, ultimately leading to improved performance, and greater scalability.

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**
    $f_\ell \leftarrow$ DynamicRoutingSelect$(F_\ell, \ell, I_\ell, I)$ ▷ f_ℓ: selected agg. function
    $O_\ell \leftarrow$ ExecuteLayer$(\ell, f_\ell, I_\ell, I)$ ▷ O_ℓ: output of layer ℓ
    Append $(\ell, f_\ell, I_\ell, O_\ell)$ to Traj
    $I_{\ell+1} \leftarrow O_\ell$ ▷ $I_{\ell+1}$: input to the next layer
end for
Back-propagation:
Global Optimization
$\mathcal{G}_{\text{global}} \leftarrow$ ComputeGlobalGradient(S, Traj) ▷ Compute global gradient
$\mathcal{S}_{\text{global}} \leftarrow$ GlobalGradientUpdate$(\mathcal{G}_{\text{global}}, \text{Traj})$ ▷ $\mathcal{S}_{\text{global}}$: Update workflow in global level
Layerwise Optimization
for each layer ℓ in reverse(L) **do**
    $\mathcal{G}^t_{local,\ell} \leftarrow$ ComputeLocalGradient$(\ell, f_\ell, \text{Traj}, \mathcal{L}_{\text{global}})$ ▷ Compute local gradient
    if momentum_needed **then**
        $\mathcal{S}_{\text{local}} \leftarrow$ LocalGradientUpdate$(\ell, f_\ell, \mathcal{G}^t_{local,\ell}, \mathcal{S}_{\text{global}})$ ▷ $\mathcal{S}_{\text{local}}$: Update layer-wise workflow
    else
        $\mathcal{G}^t_{local,\ell'} \leftarrow$ ApplyMomentum$(\ell, \text{Traj}, \mathcal{G}^t_{local,\ell}, \mathcal{G}^{t-1}_{local,\ell})$ ▷ $\mathcal{G}^t_{local,\ell'}$: Adjusted gradient
        $\mathcal{S}_{\text{local}} \leftarrow$ LocalGradientUpdate$(\ell, f_\ell, \mathcal{G}^t_{local,\ell'}, \mathcal{S}_{\text{global}})$ ▷ $\mathcal{S}_{\text{local}}$: Update layer-wise workflow
end for
return (F_ℓ, Traj) ▷ Return updated F_ℓ

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

$\mathcal{G}_{local,\ell} \leftarrow$ ComputeLocalGradient$(\ell, f_\ell, \text{Traj}, \mathcal{G}_{\text{global}})$ ▷ Compute local gradient in layer ℓ
$\mathcal{S}_{\text{local}} \leftarrow$ LocalGradientUpdate$(\ell, f_\ell, \text{G}_{local,\ell}, \text{S}_{\text{global}})$: ▷ $\mathcal{S}_{\text{local}}$: Update layer-wise workflow
for $k \leftarrow 1$ to 3 **do** ▷ Attempt up to 3 updates
    $f'_\ell \leftarrow$ LocalGradientUpdate$(\ell, f_\ell, \text{G}_{local,\ell}, \text{S}_{\text{global}})$
    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 $\mathcal{S}_{\text{global}}$

A.2.1 Prompt Template for Language Loss Function

You are a helpful AI assistant. You will use your math skills to verify the answer.
You are given:
1. A problem: {problem}
2. Reply with the answer to the problem: {final_answer}
3. A ground truth answer: {solution}

Please do the following:
Extract the answer in the reply: "The answer is <answer extracted>".
Check whether the answer in the reply matches the ground truth answer.
After everything is done, please choose and only output a reply from the following options:
1. "The answer is correct."
2. "The answer is approximated but should be correct."
3. "The answer is incorrect. Correct Answer: <ground truth answer></ground truth answer> | Answer extracted: <answer extracted></answer extracted>."
4. "The reply doesn't contain an answer."

A.2.2 Loss with ground truth and score:

Evaluate the following creative writing piece based on the provided task requirements.
Task Description: `{task_prompt}`
Creative Writing Output: `{finalized_text_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 the text align with the original task 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]

A.2.3 Prompt Template for Gradient Back-propagation

Task Description:
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.
Instructions:
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 subtask. Your output should be concise and only structured like this output format: `<output_format>{example_global_loss_format}</output_format>`.
Notice:
All analyses and suggestions should be based on a general level rather than providing very targeted suggestions for this specific task. All needed information for global optimization are provided:`{initial_solution}` For this global optimization, consider the following:
1. Final Result Evaluation: `<final result>` to determine if the task failed.
2. Solution Comparison:Compare the `<canonical solution>` and `<generated solution>`:
- Is the logic in the `<generated solution>` aligned with the `<canonical solution>`?
- Where is the gap between the analysis and the standard answer?
- Pinpoint specific issues in the `<generated solution>` that contributed to the failure.
- Write your findings into the `'global_analysis'` section of the `<output_format>`.
3. Block Input and Output Analysis:
Based on the `<task description>`, analyze the `<workflow trajectory>` to:
- Do not compare the output of each block with `<canonical solution>`. Instead, analyze which block the problem occurred.
- Examine the `block_input` and `block_output` of each block.
- Identify which block (or blocks) caused the task to fail.
- Identify inefficiencies or redundancies in the processing of the `<workflow trajectory>`.
- Document these optimization suggestions in the `'structure_suggestion'` section of the corresponding block in the `<output_format>`.
- Review the `block_description` of these blocks from `<workflow trajectory>`. If any modifications are necessary, provide suggestions and document them in the `'prompt_suggestions'` section of the corresponding block in the `<output_format>`. If modifications are not necessary, please don't give any extra suggestion.
4. Node-Level Analysis within Blocks: Based on the block(s) identified in the previous step, conduct a detailed analysis of the `node_input` and `node_output` for each node within the problematic block(s) from `<workflow trajectory>`:
- Evaluate whether the team collaboration structure or workflow within the block is effective.
- Propose specific adjustments to the team collaboration structure, if required.
- Document these optimization suggestions in the `'structure_suggestion'` section of the corresponding block in the `<output_format>`.

A.2.4 Layer Optimizer

You will evaluate the provided information for a specific block of the 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
1. Block Name: <block_name> {block_name} </block_name> 2. Global Loss Feedback: <global loss feedback>
{global_loss_feedback} </global_loss_feedback>. (This is feedback for the entire workflow in the global optimization stage. Use it as a reference, but base final modification suggestions on the best optimization solution for each layer.)
3. You should give feedback based on this blocksLog mentioned structure. Blocks Log is a record of the running track of the entire workflow when executing this task. It includes the architecture of the entire workflow, every node's input and output, and important information about all blocks and nodes. Blocks Log: {blocks_log}.
4. canonical solution of this task: <canonical solution> {canonical_solution} </canonical_solution>
5. Current task description: <current_task_ description> {task_prompt} </current_task_description>
Notice:
1. Evaluate Each Node:
- Check the 'input_variables' for each node to ensure they are valid and consistent. Valid sources for input variables include:
- Known state variables available in the workflow:
- "task_data", "task_prompt","task_id".
- Explanation:
- "task_data": Detailed task data including ID, prompt, and solutions (rarely used due to verbosity).
- "task_prompt": Description of the current task.
- "task_id": ID of the current task.
- Outputs of preceding nodes in the block, referenced by their node names (e.g., calculation_expert1_output refers to the 'node_output' of the node named calculation_expert1).
- If 'block_name' is 'ProblemSolveBlockX', an additional variable 'math_model' is available as the output of the 'ProblemAnalysisBlockX' for calculate tasks.
- When suggesting prompt modifications for a node:
- Include the updated 'prompt_template' with specific, clear instructions.
- Explicitly list all 'input_variables' along with their sources.
2. Propose Structural Changes:
- Suggest adding or removing nodes if necessary.
- For added nodes, specify:
- 'node_name': The name of the node.
- 'agent': The agent to be used by the node.
- 'Output format': The expected output format (e.g., math tool, text, number).
- 'prompt_template': The complete prompt for the node. If it contains curly braces, they must be escaped
- 'variable_sources': A dictionary specifying all input variables and their sources.
- 'constraints': The usage context and purpose of the node.
- Specify changes to node connections, including:
- 'from' and 'to' connections for new nodes.
- Impacts on other nodes, including updates to their 'input_variables' if necessary.
- Clearly identify the new 'entry_node' and 'end_node' after modifications. Each block has only one entry point and one end_node.
- Please ensure that each node has subsequent nodes connected to it to form an edge, except end_node.
- Maximal add 3 nodes.
- Include 'all_edges_now' and 'all_nodes_now' to provide a clear list of all edges and nodes in the updated block structure.

3. Impact on Other Nodes:
- Maintain logical consistency and alignment with the workflow's goals.
4. Incorporate Available Agents:
- Use the list of available agents as references for potential additions: {available_agents}.
- Refer to each agent's 'constraints' to determine effective usage.
- Ensure agents can be modified as necessary to better align with the workflow's structure, including updates to 'prompt_template', 'input_variables', 'variable_sources', or even the creation of new agents tailored to this block.
5. Dynamic Block ID and Naming:
- Assign a unique 'block_id' from the pre-calculated 'new_block_id': {new_block_id}.
- Name the block in the format '{block_name}X', where 'X' corresponds to the new 'block_id'.
6. Block Structure Description:
- Include two descriptions for the block:
- 'block_structure_description': A high-level overview of the block's purpose and role in the workflow.
- 'block_structure_description_details': A detailed explanation of the block's internal structure, including:
1. The nodes included in the block.
2. The connections (edges) between these nodes.
3. Each node's specific responsibilities.
4. How the block processes inputs and generates outputs. - Ensure both descriptions are concise, clear, and aligned with the actual block structure.
7. provided <canonical solution> and <test cases>:
- The block we are currently providing is only a part of the entire workflow, and it is possible that the failure to complete the task was not caused by this block. Therefore, please avoid over-optimization.
- Our team focuses on the entire dataset rather than a specific task. Please avoid overfitting during the refinement of suggestions and ensure that the feedback remains generalized.
- The <canonical solution> we provide is the final correct answer for this task, and the <test cases> are the test cases generated after running the entire workflow, provided for reference only. Since the block we are providing is just one part of the workflow, it is possible that the failure is not attributable to this block.
8. Output Format:
- Please provide all suggestions in the following JSON format: {layerwise_loss_format}
- Don't use an arrow to connect two nodes to represent an edge!

Figure 4: Team Structure Examples with optimization