AgentSlimming: Towards Efficient and Cost-Aware Multi-Agent Systems

Yulang Chen[1,2*] **Haoxuan Peng**[3*] **Jinyan Liu**[4*] **Zichen Wen**[1]
Dongrui Liu[5] **Linfeng Zhang**[1†]

[1]Shanghai Jiao Tong University [2]Nanjing University [3]The University of Sydney
[4]Nanjing University of Aeronautics and Astronautics
[5]Shanghai AI Laboratory

Abstract

Large Language Model-based Multi-Agent Systems (MAS) have demonstrated remarkable capabilities in complex tasks. However, manually designing optimal communication topologies is labor-intensive, while automated expansion methods often result in bloated structures with redundant agents, leading to excessive token consumption. To address this problem, we introduce **AgentSlimming**, a plug-and-play compression framework for graph-structured multi-agent workflows. Motivated by pruning and quantization in neural networks, AgentSlimming compresses workflows by first estimating the importance score of each agent with a hybrid mechanism, and then removes redundant agents or replaces them with low-cost ones, where each operation is validated using a baseline-anchored acceptance rule to prevent performance collapse. Experiments show that AgentSlimming reduces average token cost by up to 78.9% with negligible performance degradation, and sometimes even improves accuracy, achieving a strong Pareto-optimal trade-off between cost and quality. *Our code is publicly available at* `https://github.com/CitrusYL/AgentSlimming`.

1 Introduction

The paradigm of Multi-Agent Systems (MAS) has shifted from manually crafting static prompts to orchestrating dynamic collaborations among specialized agents. Frameworks such as AutoGen (Wu et al., 2023) and ADAS (Hu et al., 2025) have shown that decomposing complex problems into sub-tasks can improve performance. Recent innovations, such as MetaGPT (Hong et al., 2024) and AFlow (Zhang et al., 2025c), have revolutionized this field by reformulating workflow generation as a search problem. By leveraging Monte Carlo Tree Search (MCTS), AFlow can automatically navigate the vast space of agent interactions to discover highly effective reasoning topologies.

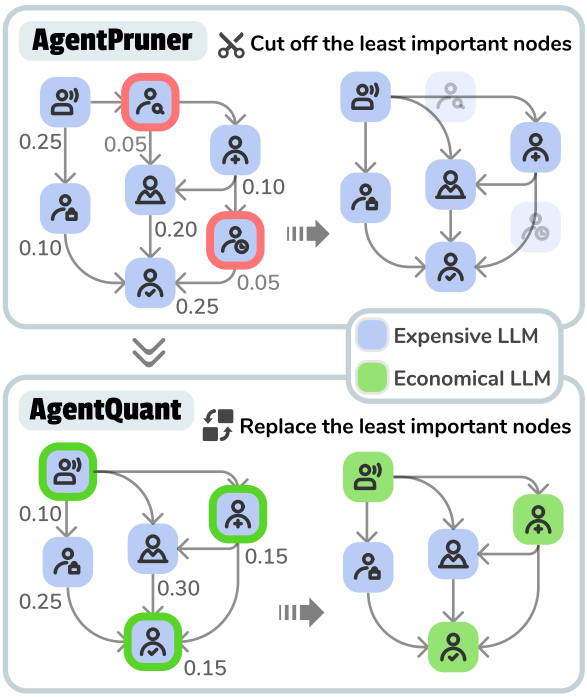

Figure 1: **Overview of the pruning and quantization concepts in AgentSlimming.** The process begins with a workflow that is initially high-performance but computationally expensive. Then, a *hybrid importance evaluation mechanism* is utilized to calculate each agent node's importance score (*i.e.,* the float value), which guides *pruning* and *quantization* in AgentSlimming.

However, this automated discovery process operates under an "unconstrained resources" assumption. The resulting workflows, while accurate, tend to be computationally bloated. They often feature dense, redundant connectivity and uniformly employ the most capable and expensive LLMs, e.g., GPT-4 (OpenAI, 2023), as agent nodes for every sub-task (Li et al., 2023; Qian et al., 2024). This leads to two critical bottlenecks: **redundant communication**, where agents exchange repetitive or low-value information, and **excessive computa-**

*Equal contribution.
†Corresponding author (zhanglinfeng@sjtu.edu.cn).

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 30064–30086
July 2-7, 2026 ©2026 Association for Computational Linguistics

tional cost, where simple sub-tasks consume high-value resources. As the number of agents and interaction turns grows, a quadratic increase in token consumption renders these systems difficult to scale.

To solve this problem, we propose AgentSlimming, a framework that can be directly applied to any MAS and workflows to reduce their execution costs. Motivated by the concepts of pruning (Han et al., 2015; Frankle and Carbin, 2019), quantization (Jacob et al., 2018; Dettmers et al., 2022), and finetuning (Howard and Ruder, 2018) in traditional neural network compression, we define our pruning as the removal of an agent node, and semantic quantization as replacing the LLM associated with an agent node with a lower-cost model. AgentSlimming introduces AgentPruner, AgentQuant and AgentTuner as tools towards lightweight MAS. As shown in Figure 1, AgentPruner identifies the least important agent nodes in the MAS and removes them, while AgentQuant replaces the underlying LLMs of the remaining lower-importance nodes with more cost-effective alternatives. When high-cost, high-precision models (e.g., GPT-4) are considered unnecessary for a given role in MAS, they are dynamically replaced by cost-effective, lightweight surrogates (e.g., GPT-4o-mini) to reduce the inference costs. After AgentPruner and AgentQuant, AgentTuner is optionally employed to recover performance loss and further optimize.

The proposition of AgentSlimming introduces a crucial problem for MAS compression, *i.e.,* how to determine the importance of each node in MAS. To tackle this challenge, we introduce a hybrid importance evaluation mechanism. Generally, a node's value is determined by both its *structural position* and *functional contribution*. Based on this observation, we consider the following three indicators: Degree Centrality (Freeman, 1978) and Betweenness Centrality (Brandes, 2001) to capture the topological structure in the graph of MAS, and an Approximate Shapley value (Shapley, 1953; Lundberg and Lee, 2017) for functional contribution. Specifically, we adopt a Leave-One-Out (LOO) estimation strategy to calculate the marginal contribution of each node against the complete topology, serving as a computationally efficient proxy for the exact Shapley value. These rankings are fused via Reciprocal Rank Fusion (RRF) (Cormack et al., 2009) to generate a robust importance score.

Guided by this score, AgentSlimming employs an iterative greedy strategy: we progressively prune and quantize agent nodes starting from the least important ones. After each operation, we re-evaluate the workflow against a performance threshold and re-calculate the RRF scores, ensuring the optimization dynamically adapts to the changing graph structure until the efficiency limit is reached. While the pruned and quantized workflows typically achieve a superior Pareto frontier (Deb et al., 2002; Chen et al., 2024) between cost and accuracy, AgentSlimming further incorporates a MCTS fine-tuning mechanism that optimizes the streamlined graph rather than just accepting the compressed policy. Empirically, the resulting workflows are consistently more cost-efficient than AFlow, often achieving strictly superior performance in both accuracy and cost. In terms of average execution cost per problem (USD), AgentSlimming demonstrates significant gains across diverse benchmarks.

For instance, on GSM8K, AgentSlimming matches AFlow's score (95.5) while reducing the cost by **78.8%**. On code generation and complex reasoning tasks like MBPP and LiveCode, AgentSlimming achieves a "dual victory": it improves accuracy while reducing costs by **71.7%** and **78.9%**, respectively. Even in cases where AFlow retains a marginal score advantage (e.g., HotpotQA: 77.3 vs. 77.0), AgentSlimming achieves this at a substantially lower cost (27.4% reduction), indicating that our pipeline reliably locates the workflow at a more favorable operating point on the cost–quality Pareto frontier. Our contributions can be summarized as follows:

- We introduce AgentSlimming, a training-free framework that integrates automated agentic workflow exploration with a novel pruning and semantic quantization pipeline.
- We propose a novel iterative multi-metric optimization algorithm that integrates topological and functional importance via RRF to accurately identify redundant components.
- Extensive evaluations on eight benchmarks show that AgentSlimming achieves a striking reduction in token costs of up to **78.9%**.

2 Related Work

2.1 Agentic Workflows

LLM-based systems can be broadly characterized into two paradigms: *agentic workflows* and *autonomous agents*. The former executes tasks

through predefined multi-step pipelines with repeated LLM invocations, whereas the latter emphasizes dynamic decision-making and planning through interaction and feedback. Recent work has made notable progress in language-driven decomposition and collaboration (Zhuge et al., 2025), data-science agents (Hong et al., 2025), tool-enabled mobile agent teams (Zhang et al., 2024), and open-ended embodied exploration (Wang et al., 2024a). Compared to autonomous agents that often require environment-specific action spaces and decision patterns, workflows can more easily incorporate human domain expertise and improve through iterative refinement, making them particularly amenable to automated construction and optimization.

2.2 Multi-Agent Evolving

Many effective workflows are still developed primarily through manual discovery and engineering practice. At the general level, common transferable reasoning recipes include chain-of-thought, self-consistency, self-refine and structured self-collaboration (Wei et al., 2022; Wang et al., 2023; Madaan et al., 2023; Wang et al., 2024b). At the domain level, multi-step procedures are organized into reusable pipelines, such as code generation and debugging (Hong et al., 2024; Ridnik et al., 2024; Zhong et al., 2024), data analysis and visualization (Xie et al., 2024; Ye et al., 2024; Li et al., 2024a; Zhou et al., 2023), mathematical reasoning (Xu et al., 2024), and planning-style search for problem solving (Nori et al., 2023; Zhou et al., 2024).

However, manual design cannot cover the combinatorial space across domains, which motivates automated agentic optimization. One line of work optimizes local instructions or components within a fixed backbone (Fernando et al., 2024; Yüksekgönül et al., 2024; Yang et al., 2024; Khattab et al., 2024) or tunes inference-time strategies (Saad-Falcon et al., 2024). Another line goes further by optimizing the end-to-end workflow structure, including automatic generation of code-represented workflows (Li et al., 2024b), viewing agent systems as optimizable graphs (Zhuge et al., 2024), and improving system designs in code space via meta-agents (Hu et al., 2025). Zhang et al. (2025c) similarly adopts code-based representations, combining finer-grained abstractions (named nodes/operators) with MCTS to leverage execution feedback and tree-structured experience for efficient workflow structure search.

2.3 Graph-Structured Orchestration, Pruning, and Cost-Aware Compression

Beyond prompt engineering, many agentic systems are best viewed as executable graphs, where nodes represent specialized modules and edges encode information flow. Recent studies focus on topology learning and pruning for improved efficiency and robustness (Zhang et al., 2025a,b; Boyi et al., 2025). These approaches also explore pruning at different granularities, including message-level pruning under bandwidth constraints (Mao et al., 2020), dynamic agent elimination for token efficiency (Wang et al., 2025), and progressive pruning that blends heuristics with execution experience (Zhang et al., 2025d). Existing methods typically start from (near) fully connected interactions and primarily prune edges or communication channels. Action selection for pruning and replacement can draw on heuristic importance measures from network analysis (Freeman, 1978) or contribution-based formulations such as Shapley values (Shapley, 1953), which usually require approximation for tractability. In addition, model compression and quantization reduce inference cost while preserving quality (Frantar et al., 2022; Xiao et al., 2023; Lin et al., 2024), complementing structural optimization at the workflow level.

2.4 Search-Based Optimization for Workflows

Search provides a natural mechanism for exploring large workflow design spaces. MCTS (Coulom, 2006) and its variant UCT (Kocsis and Szepesvári, 2006) enable effective exploration through sampled evaluation and progressive expansion, and has been applied to planning and reasoning in language agents (Zhou et al., 2024) as well as workflow structure optimization (Zhang et al., 2025c). Following this trajectory, we focus on optimizing DAG workflows under dependency constraints, integrating structure search with node-level pruning and cost-driven node replacement (semantic quantization).

3 Methodology

3.1 Problem Formulation

We formulate the optimization of multi-agent systems as a discrete structural search problem over a directed graph, with an explicit trade-off between task performance and execution cost.

Workflow as a directed graph. A multi-agent workflow is represented as a directed graph $\mathcal{G} =$

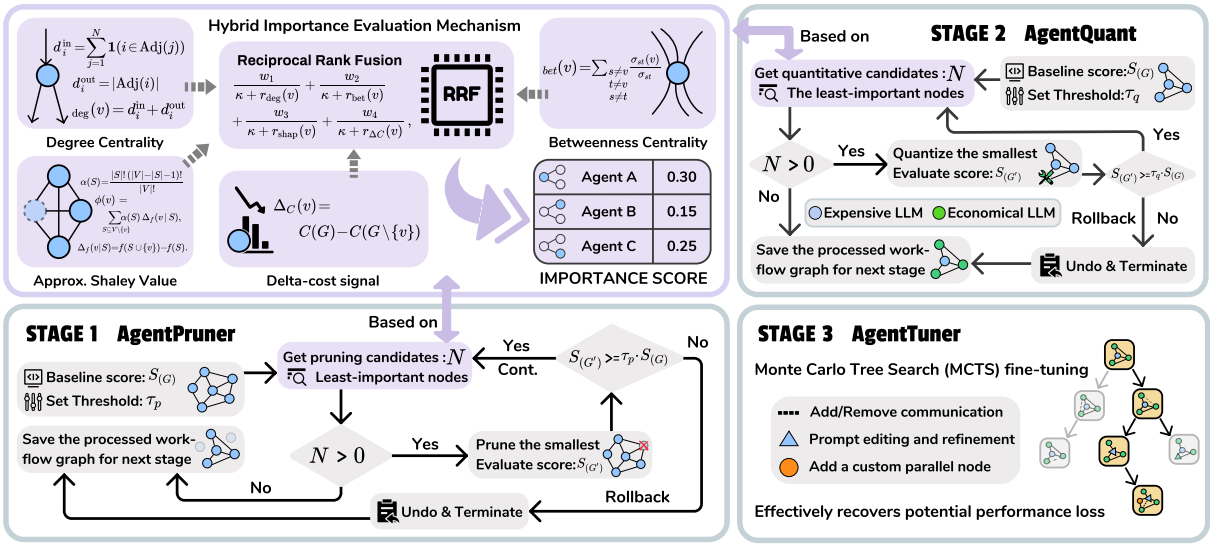

Figure 2: **Illustration of AgentSlimming.** To identify redundancy, AgentSlimming computes four distinct rankings of each agent node: *Degree Centrality*, *Betweenness Centrality*, *Cost Comparison* and *Approximate Shapley value*. Both pruning and quantization select candidate nodes based on the computed importance scores, ranking nodes from least to most important and optimizing the lowest-ranked candidate first. After each operation, a re-evaluation is performed. If the score drops below an acceptable threshold, a rollback mechanism is triggered to revoke the current operation and immediately terminate this phase, thereby preserving superior performance.

$(\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ denotes an executable operator (e.g., an LLM call), and each edge $e \in \mathcal{E}$ specifies an information dependency. Given a validation set $\mathcal{D}_{\text{val}}$, executing $\mathcal{G}$ yields an average task score $S(\mathcal{G})$ and an average execution cost $C(\mathcal{G})$:

$$S(\mathcal{G}) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{x \in \mathcal{D}_{\text{val}}} \text{score}(\mathcal{G}, x),$$
$$C(\mathcal{G}) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{x \in \mathcal{D}_{\text{val}}} \text{cost}(\mathcal{G}, x). \quad (1)$$

Optimization Objective. Our goal is to maximize performance under a cost budget B. When multiple topologies achieve comparable performance, we prioritize the one with the minimum cost:

$$\max_{\mathcal{G}} \; S(\mathcal{G}) \quad \text{s.t.} \quad C(\mathcal{G}) \leq B. \quad (2)$$

3.2 The AgentSlimming Pipeline

As illustrated in Figure 2, our framework operates through a three-stage pipeline: *AgentPruner (Structural Pruning)*, *AgentQuant (Semantic Quantization)*, and *AgentTuner (MCTS Fine-tuning)*. This pipeline progressively compresses the workflow while maintaining its reasoning capabilities.

Hybrid Importance Evaluation To guide both pruning and quantization, we introduce a hybrid importance evaluation mechanism. For any node $v \in \mathcal{V}$, we compute four complementary signals using a small probe dataset $\mathcal{D}_{\text{probe}} \subset \mathcal{D}_{\text{val}}$. To evaluate the sensitivity of our method to the probe dataset size, we further conduct additional experiments on HotpotQA, with details provided in Appendix F.

1. Degree and Betweenness Centrality Signals (Topological Priors) We capture the structural significance of a node using graph centrality metrics.

- **Degree Centrality Signal (s_{deg}):** It is derived from the node's local connectivity (in-degree and out-degree). Nodes with weaker structural connectivity are assigned larger pruning likelihood.

- **Betweenness Centrality Signal (s_{bet}):** Measures the node's role as an information bridge. It is calculated as:

$$s_{\text{bet}}(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (3)$$

where σ_{st} is the total number of shortest paths from s to t, and $\sigma_{st}(v)$ is the number of those paths passing through v.

2. Approximate Shapley value Signal (Functional Contribution) To quantify the semantic

contribution of node v, we adopt the Shapley value concept from cooperative game theory. Since exact computation is NP-hard, we employ an efficient **Leave-One-Out (LOO)** approximation. The marginal contribution $\hat{\phi}(v)$ is estimated by the performance drop when v is removed (or replaced):

$$\hat{\phi}(v) \approx \Delta_S(v) = S(\mathcal{G}) - S(\mathcal{G} \setminus \{v\}), \quad (4)$$

where $\mathcal{G} \setminus \{v\}$ denotes the workflow after removing node v (and patching connections). A smaller $\hat{\phi}(v)$ indicates a lower contribution to task completion.

3. Delta-Cost Signal (Economic Potential) We estimate the potential cost saving $\Delta_C(v)$ on $\mathcal{D}_{\text{probe}}$:

$$\Delta_C(v) = C(\mathcal{G}) - C(\mathcal{G} \setminus \{v\}), \quad (5)$$

where $C(\cdot)$ denotes the monetary API cost, measured in USD per problem, incurred when executing the workflow on $\mathcal{D}$probe. Specifically, it is computed based on the actual API billing rate and the total token usage (input and output tokens) of all model calls within the workflow. Nodes with higher $\Delta_C(v)$ are prioritized for pruning or quantization to maximize efficiency gains.

Rank Fusion To robustly combine these heterogeneous signals, we employ Reciprocal Rank Fusion (RRF). Let $r_m(v)$ be the rank of node v under metric $m \in \{\text{deg}, \text{bet}, \text{shap}, \Delta C\}$. The fused score is calculated as:

$$\text{RRF}(v) = \sum_m \frac{w_m}{\kappa + r_m(v)}, \quad (6)$$

where κ is a smoothing constant (set to $\max\{10, |\mathcal{V}|\}$) and w controls the relative weight of each metric. This fused score identifies nodes that are simultaneously structurally peripheral, functionally redundant, and computationally expensive.

3.2.1 Iterative Optimization Process

Stage 1: AgentPruner (Structural Pruning) Let $\mathcal{G}_{\text{base}}$ denote the initial high-performance workflow graph. We employ an iterative greedy strategy to prune nodes. In each iteration, we rank nodes by $\text{RRF}(v)$ and evaluate the top candidates. A pruning operation is accepted if the performance degradation is within a tolerance threshold τ_p:

$$S(\mathcal{G}_p) \geq \tau_p \cdot S(\mathcal{G}_{\text{base}}), \quad (7)$$

where $\mathcal{G}_p$ is the pruned graph and $\tau_p \in (0, 1)$ is a predefined threshold. We evaluate the Top-1 candidate first and then the remaining candidates in order if necessary. If neither candidate is accepted, the stage terminates and the workflow rolls back to the last accepted state. **Graph Surgery:** To preserve executability after removing node v, we perform edge patching to maintain the graph's connectivity. For every pair of predecessor $s \in \text{In}(v)$ and successor $t \in \text{Out}(v)$, we add a direct edge $(s \to t)$ if one does not already exist, where self-loops and duplicate edges are disallowed. Each accepted pruning step is logged with the probe signals, fused ranks, and the resulting workflow artifact.

Stage 2: AgentQuant (Semantic Quantization) $\mathcal{G}_p$ refers to the pruned workflow graph obtained from Stage 1. The quantization process operates on this sparse structure to further reduce computational costs. We define *Semantic Quantization* as a model substitution strategy. For a selected node v, we replace its high-cost LLM with a cost-effective surrogate $\pi(v)$, yielding a quantized graph $\mathcal{G}_q$. Candidate ranking follows the same RRF mechanism, where the Shapley signal is instantiated by evaluating the performance impact of the model substitution. We also employ a greedy strategy to quantize nodes. A quantization operation is accepted if the performance degradation is within a tolerance threshold τ_q:

$$S(\mathcal{G}_q) \geq \tau_q \cdot S(\mathcal{G}_p), \quad (8)$$

where $\mathcal{G}_q$ is the quantized graph and $\tau_q \in (0, 1)$ is a predefined threshold. This step drastically reduces token costs for non-critical reasoning steps. We employ a "greedy + rollback" optimization strategy, as global combinatorial search on large graphs is computationally prohibitive. Although this risks missing complex multi-node synergies, we mitigate this limitation by using approximate Shapley values and relaxed thresholds to balance efficiency with effective exploration.

Stage 3: AgentTuner (Adaptive MCTS Fine-tuning) To address the variance in compression sensitivity across different tasks, we adapt the MCTS exploration from AFlow (Zhang et al., 2025c) for our compressed workflow. This phase is selectively applied and focuses on localized refinements, such as prompt adaptation and parameter tuning, to recover from potential degradation. Critically, this fine-tuned configuration is adopted only

if it improves upon the initial compressed version, thereby ensuring consistent and robust performance gains.

4 Experiments

4.1 Setup

Overview. We evaluate our method on eight public benchmarks. To ensure a fair comparison, we strictly adhere to AFlow's configuration, adopting its datasets, splits, and sampling methods. Additionally, we incorporate three challenging benchmarks to test robustness.

Datasets. Our evaluation suite consists of two categories: (1) **Standard Benchmarks.** We utilize the complete AFlow (Zhang et al., 2025c) suite, including GSM8K (Cobbe et al., 2021), MBPP (Austin et al., 2021) (full sets), as well as HotpotQA (Yang et al., 2018) and DROP (Dua et al., 2019) (randomly sampled 1,000-instance subsets). For MATH (Hendrycks et al., 2021), we follow the specific subset of 617 Level-5 problems across four categories. These datasets employ a 1:4 validation/test split. (2) **High-difficulty Benchmarks.** To assess performance on complex reasoning and coding tasks, we incorporate AIME (Mathematical Association of America; Art of Problem Solving) , MuSiQueAns (Trivedi et al., 2022), and LiveCode (Jain et al., 2025). These datasets are partitioned using a 3:7 validation/test split. Across both categories, our evaluation suite collectively covers three core competencies: mathematical reasoning, coding, and question answering, ensuring a comprehensive assessment of agent system performance.

Baselines. We compare AgentSlimming against a diverse set of baselines: *(I) Manual Prompting Strategies:* Standard Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (SC-CoT) (Wang et al., 2023), Self-Refine (Madaan et al., 2023), and LLM-Debate (Du et al., 2023). *(II) Automated Workflow Optimization:* ADAS (Hu et al., 2025) and AFlow (Zhang et al., 2025c). *(III) Cost-Aware AFlow:* We implement a modified AFlow variant that explicitly integrates cost comparisons into its selection mechanism during the search phase, directly competing on cost efficiency.

Details. We employ GPT-4.1-mini (OpenAI, 2025) as the high-precision model and GPT-4.1-nano (OpenAI, 2025) as the cost-effective surrogate for quantization, both accessed via the OpenAI API. For workflow-based optimization, we use GPT-4.1-mini to maintain computational efficiency. All models are accessed with temperature set to 0 to ensure reproducibility. Full hyperparameter configurations are provided in Appendix C. We further extend our study with cross-architecture experiments on the Qwen3 series, cross-dataset validation on GSM8K and MATH, and additional experiments based on ADAS, which demonstrate that AgentSlimming is not tied to AFlow-specific design choices; detailed results are provided in Appendix G.

4.2 Main Results

Superior Cost-Performance Efficiency. The main experimental results are summarized in Table 1 and Table 2. Overall, AgentSlimming consistently discovers workflows that are more cost-efficient than AFlow. In many cases, our method achieves *strict dominance*, surpassing the baseline in both accuracy and execution cost. Here, we report the average cost in USD per problem.

On standard benchmarks, AgentSlimming maintains competitive performance while significantly reducing computational overhead. For instance, on GSM8K, it matches AFlow's accuracy (95.5) but reduces the average cost from 4.38×10^{-3} to 9.30×10^{-4}, a **78.8%** cost reduction. On MBPP, AgentSlimming improves the score from 73.3 to 77.9 while simultaneously lowering the cost from 2.58×10^{-3} to 7.30×10^{-4} (**71.7%** reduction). On datasets where AFlow attains a slightly higher raw score, such as HotpotQA, AgentSlimming achieves comparable performance at a substantially lower cost, reducing the average cost by 27.4%.

The advantage of AgentSlimming is even more pronounced on high-difficulty benchmarks. On LiveCode, it increases the score from 55.3 to 61.7 while reducing cost by nearly an order of magnitude (from 1.17×10^{-2} to 2.47×10^{-3}, a **78.9%** reduction). Similarly, on MuSiQueAns, AgentSlimming improves accuracy from 84.5 to 89.3 with a **23.2%** reduction. These results indicate that AgentSlimming reliably positions the discovered workflows at more favorable operating points on the cost–quality Pareto frontier, particularly for complex reasoning and programming tasks.

4.3 Analysis

We conducted a comprehensive analysis to identify the sources of efficiency gains in AgentSlimming and validate its practical feasibility.

Method	MATH		GSM8K		HotpotQA		MBPP		DROP	
	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)
Ours	73.9	6.88e-3 ↓42.7%	**95.5**	9.30e-4 ↓78.8%	77.0	5.55e-3 ↓27.4%	**77.9**	7.30e-4 ↓71.7%	**81.7**	9.50e-4 ↓38.7%
AFlow	**74.8**	1.20e-2	**95.5**	4.38e-3	**77.3**	7.64e-3	73.3	2.58e-3	78.7	1.55e-3
LLM-Debate	74.5	1.47e-2	93.5	3.57e-3	73.4	9.47e-3	75.1	6.22e-3	75.6	2.94e-3
ADAS	69.0	1.15e-2	94.9	4.79e-3	65.4	4.72e-2	75.4	4.31e-3	78.6	3.07e-3
CoT	62.9	6.03e-4	90.5	2.48e-4	57.5	7.11e-4	73.6	2.80e-4	75.9	3.75e-4
SC-CoT	65.3	3.67e-3	90.2	1.71e-3	61.7	4.48e-3	75.6	2.12e-3	75.3	2.37e-3
Self-Refine	65.5	1.01e-3	89.8	5.60e-4	57.9	1.50e-3	74.5	5.56e-4	75.0	7.91e-4

Table 1: **Performance and inference cost comparison across standard benchmarks.** Cost denotes the average API expense ($/prob.). **Bold** indicates the best results among **workflow-based methods** (excluding simple prompting baselines). Green percentages indicate the relative cost reduction compared to AFlow, highlighting how much inference budget is saved while maintaining competitive accuracy.

Method	AIME		LiveCode		MusiqueAns	
	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)
Ours	65.7	1.35e-2 ↓19.2%	61.7	2.47e-3 ↓78.9%	**89.3**	8.24e-3 ↓23.0%
AFlow	**67.1**	1.67e-2	55.3	1.17e-2	84.5	1.07e-2
LLM-Debate	56.8	2.86e-2	**67.8**	1.67e-2	45.3	1.62e-2
Cost-aware AFlow	61.8	1.20e-2	57.9	8.41e-3	83.6	1.04e-2
ADAS	62.3	1.42e-2	52.1	9.01e-3	77.2	1.62e-2
CoT	53.4	2.99e-3	46.4	1.25e-3	73.9	1.11e-3
Self-Consistency CoT	59.7	1.41e-2	47.2	7.93e-3	74.9	6.89e-3
Self-Refine	54.1	4.08e-3	49.1	2.30e-3	72.8	2.30e-3

Table 2: **Performance and inference cost comparison across high-difficulty benchmarks.** Cost denotes the average API cost ($/prob.). **Bold** indicates the best results among **workflow-based methods** (excluding simple prompting baselines). Green percentages indicate the relative cost reduction compared to AFlow.

Denoising via Pruning In the initial heavy workflows, we observed redundant parallel generation components, particularly multiple `CodeGenerate` nodes and `AnswerGenerate` nodes. Our method consistently deprioritizes these redundant nodes due to their limited marginal benefit, pruning them to yield a simplified graph structure.

Strategic Heterogeneity When parallel structures remain after pruning, AgentSlimming applies non-uniform compression and retains a mixed-precision configuration. Specifically, it preserves a small number of high-capability `AnswerGenerate` nodes at full precision and quantizes the remaining parallel auxiliary nodes to lower-cost models.

Cost-Effective Arbitration via Functional Alignment The quantization frequencies indicate that AgentSlimming primarily compresses nodes whose functionality is procedure-oriented rather than generation-intensive. `ScEnsembler` and `Programmer` nodes are infrequently pruned but are frequently quantized, while `Test` nodes are consistently quantized. Although these nodes can be topologically central, they mainly implement selection, consensus aggregation, and execution-based validation. By selectively quantizing them while retaining high-capability generators, AgentSlimming preserves accuracy and strategically directs the computational budget toward core reasoning rather than auxiliary validation.

Total Evaluation Budget Analysis. To translate the structural efficiency gains into tangible economic implications, we estimate the total end-to-end evaluation budget for the test sets of each benchmark by multiplying the optimized per-problem cost by the effective sample size. Remarkably, the computational burden is minimal: standard benchmarks like MBPP and GSM8K require less than **$2.00** to evaluate the entire test set ($1.63 and $1.84, respectively). Even for computationally intensive, high-difficulty benchmarks such as AIME and HotpotQA, the total costs remain strictly affordable at $26.06 and $14.76.

Crucially, this cost-effectiveness also extends beyond inference to the optimization process itself.

Method	MATH		MBPP		LiveCode	
	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)
Baseline	74.8	1.20e-2	73.3	2.58e-3	55.3	1.17e-2
Quantization -> Pruning	76.5	3.05e-3	75.6	6.30e-4	59.8	1.99e-3
Pruning -> Quantization (Original Method)	73.9	6.88e-3	77.9	7.30e-4	61.7	2.47e-3

Table 3: **Ablation of reversing the pipeline order.** We compare the baseline, the reversed-order pipeline (*Pruning after Quantization*), and the final *Quantization after Pruning* results on MATH, MBPP, and LiveCode, which represent mathematical reasoning, program synthesis, and execution-oriented code generation, respectively. Empirically, reversing the order produces a broadly similar final cost-accuracy trade-off, with only minor differences arising from slightly different search trajectories under the greedy rollback mechanism. We nevertheless adopt the *Pruning* $\rightarrow$ *Quantization* design in our framework, because pruning removes redundant nodes early, shrinks the search space, and thus makes the subsequent quantization stage more efficient and reduces the overall search cost of the framework.

Method	AIME		DROP		MATH		MBPP	
	Acc.	Cost	Acc.	Cost	Acc.	Cost	Acc.	Cost
Baseline	67.1	1.67e-2	78.7	1.55e-3	74.8	1.20e-2	73.3	2.58e-3
Betweenness-only	63.0	1.15e-2	77.8	4.5e-4	74.0	4.17e-3	79.1	7.1e-4
Degree-only	63.6	7.89e-3	66.5	3.2e-4	73.1	5.28e-3	73.3	1.56e-3
Shapley-only	64.7	7.81e-3	65.7	3.1e-4	71.4	5.25e-3	74.4	1.50e-3
RRF (ours)	65.7 (-2.1%)	1.35e-2 (-19.2%)	78.4 (-0.4%)	4.5e-4 (-71.0%)	74.8 (0.0%)	2.54e-3 (-79.7%)	80.2 (+9.4%)	7.1e-4 (-72.5%)

Table 4: **Ablation study of the importance ranking strategies.** We compare topology-based signals (degree-only, betweenness-only) and a functional signal (Shapley-only), as well as their fusion via Reciprocal Rank Fusion (RRF), across four representative benchmarks: DROP for reading comprehension and discrete numerical reasoning, MATH and AIME for mathematical and symbolic reasoning (AIME further targeting competition-level problem solving), and MBPP for program synthesis. We report task accuracy (Acc.) and average inference cost per problem ($/prob.). The percentages in the RRF row indicate the relative change on each benchmark compared to the baseline.

As a fully training-free framework, AgentSlimming avoids the prohibitive overhead of gradient-based updates and expensive parameter tuning. Furthermore, by leveraging topological priors (e.g., degree and betweenness centrality) for importance scoring, we ensure that pruning and quantization remain computationally efficient and lightweight. To further quantify the trade-off between the one-time optimization overhead and the recurring inference-time savings, we additionally analyze the time-to-break-even of our method, i.e., how many future executions are needed to amortize the search cost through reduced per-query inference cost. We report the detailed formulation and benchmark-wise optimization cost in Appendix E.

4.4 Ablation Study

We conduct two complementary ablation studies. First, we examine the effect of pipeline order by comparing our default *Prune* $\rightarrow$ *Quantize* design with the reversed *Quantize* $\rightarrow$ *Prune* variant. As shown in Table 3, reversing the order leads to only marginal changes in the final cost–accuracy trade-off, indicating that the final Pareto frontier is driven mainly by the joint effect of pruning and quantization rather than by a specific execution order. The small differences mainly arise from slightly different search trajectories under the greedy rollback mechanism. We nevertheless adopt *Prune* $\rightarrow$ *Quantize* in the main framework, because pruning first removes redundant nodes early, shrinks the workflow graph, and reduces the search cost of the subsequent quantization stage.

We then investigate the impact of the node-importance ranking strategy used in pruning and quantization. Our full method employs a weighted Reciprocal Rank Fusion (RRF) scheme to synthesize multiple signals into a unified candidate ranking, whereas the ablated variants rely on a single metric, namely Degree-only, Betweenness-only, and Shapley-only. To isolate the effect of ranking quality from confounders such as search budget, initialization, and stopping criteria, we keep the experimental protocol consistent across stages: for pruning, all variants start from the original workflow; for quantization, each variant inherits the best pruned graph derived from its corresponding prun-

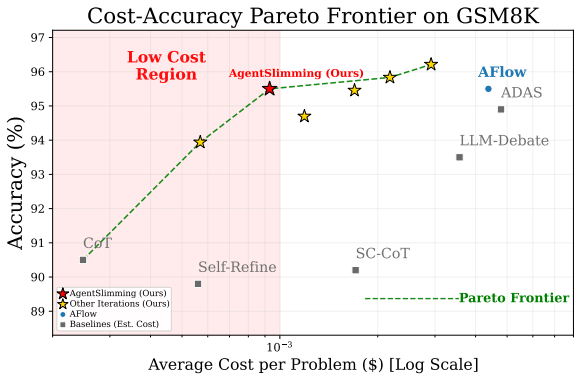

Figure 3: **Cost-Accuracy Pareto Frontier on the GSM8K dataset.** The x-axis represents the average inference cost per problem (USD), and the y-axis denotes accuracy. Data points for our method correspond to varying iteration rounds. These configurations consistently align with the Pareto frontier, demonstrating optimal cost-quality trade-offs compared to baselines.

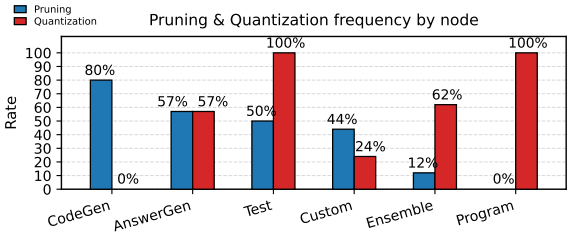

Figure 4: **Node-type specific compression rates.** We report the ratio of nodes processed by each compression strategy across six categories. Blue bars indicate structural removal (Pruning), while red bars indicate model substitution (Quantization). The variation across categories highlights the adaptive nature of our hybrid importance evaluation.

ing strategy and applies the same metric for node selection. Both stages use identical validation subset sizes and baseline-anchored acceptance thresholds. Table 4 summarizes the resulting performance. We observe that single-metric strategies exhibit divergent cost–quality trade-offs across tasks, highlighting their complementary strengths and motivating multi-signal fusion. In contrast, RRF consistently achieves a more robust balance between effectiveness and efficiency. Detailed stage-wise ablation results are provided in Appendix D.

4.5 Case Study

We analyze the pruning process of the workflow on MATH to illustrate structural simplification. Starting from a complex graph with parallel reasoning paths and refinement steps, AgentSlimming identified redundancy in the dual `AnswerGenerate` nodes and the marginal utility of the `CodeRefine` node. Concretely, probe-based importance ranking suggested that one generator path consistently dominates in contribution, while refinement rarely corrects errors relative to its token cost. Notably, pruning the parallel path diminished the utility of the downstream `ScEnsemble` node, leading to its removal (as demonstrated in Figure 5). The result is a streamlined pipeline that preserves core reasoning logic while achieving substantial cost reductions, empirically validating our topo-functional optimization strategy. The complete details of this case study are provided in Appendix B.

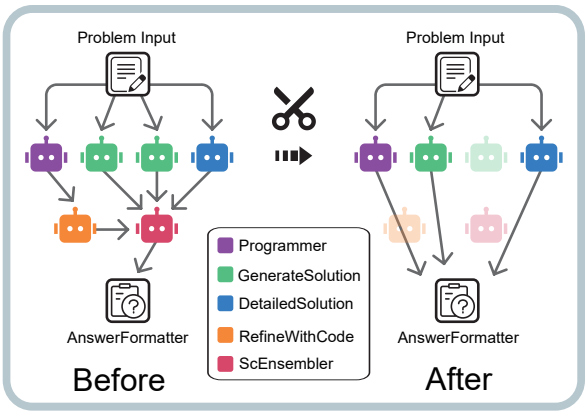

Figure 5: **Visualizing the pruning process of MATH workflow.** (*Before*) The initial workflow features complex parallel reasoning paths and an ensemble mechanism. (*After*) The streamlined workflow after AgentSlimming's pruning stage. It effectively identifies the core reasoning backbone by pruning redundant parallel paths and refinement steps, thereby reducing cost.

5 Conclusion

We introduced AgentSlimming, a unified framework that optimizes multi-agent workflows via structural pruning and semantic quantization. Guided by a novel hybrid metric combining topological and functional signals, AgentSlimming achieves up to **78.9%** cost reduction across diverse benchmarks. Importantly, this efficiency is attained with negligible impact on reasoning quality, matching or even exceeding baseline performance. Our method shows that sparse, heterogeneous topologies can effectively replace computationally redundant dense multi-agent systems, establishing a new Pareto frontier for scalable agentic collaboration.

Limitations

We evaluate GPT-4.1-series and Qwen3-series models, but do not benchmark newer frontier models.

Like AFlow, our framework currently operates at the task level to ensure fair baseline comparisons. For future work, we plan to explore instance-level dynamic adaptation, utilizing multi-agent memory to adjust workflows and compute based on query difficulty.

Acknowledgements

This work was supported by Alibaba Group through Alibaba Innovative Research Program.

Ethics Statement

We acknowledge that all authors are informed about and adhere to the ACL ARR Code of Ethics and the Code of Conduct.

Risks Our benchmarks are sourced from publicly available datasets. We cannot guarantee that they are free of socially harmful, biased, or toxic content. In addition, AgentSlimming optimizes cost by pruning and replacing nodes, which may alter the behavior of the original workflow in unexpected ways; when applied to safety-critical domains, such changes could amplify errors or reduce reliability. We used LLM-based tools only for grammar and language polishing; all technical content, experiments, and claims were written and verified by the authors.

References

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Li Boyi, Zhonghan Zhao, Der-Horng Lee, and Gaoang Wang. 2025. Adaptive graph pruning for multi-agent communication. *CoRR*, abs/2506.02951.

Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2024. Frugalgpt: How to use large language models while reducing cost and improving performance. *Trans. Mach. Learn. Res.*

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*.

Rémi Coulom. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer.

Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323.

Linton C. Freeman. 1978. Centrality in social networks conceptual clarification. *Social Networks*, 1.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Sirui Hong, Yizhang Lin, Bang Liu, and 1 others. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 19796–19821. Association for Computational Linguistics.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, and 1 others. 2024. Metagpt: Meta programming for A multi-agent collaborative framework.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339. Association for Computational Linguistics.

Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2704–2713. Computer Vision Foundation / IEEE Computer Society.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, and 1 others. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to SQL: are we fully ready? [experiment, analysis & benchmark]. *Proc. VLDB Endow.*, 17(11):3318–3331.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zelong Li, Shuyuan Xu, Kai Mei, and 1 others. 2024b. Autoflow: Automated workflow generation for large language model agents. *CoRR*, abs/2407.12821.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. 2024. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. *GetMobile Mob. Comput. Commun.*, 28(4):12–17.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Aman Madaan, Niket Tandon, Prakhar Gupta, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning agent communication under limited bandwidth by message pruning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5142–5149. AAAI Press.

Mathematical Association of America. Maa invitational competitions: American invitational mathematics examination (aime). `https://maa.org/maa-invitational-competitions/`.

Harsha Nori, Yin Tat Lee, Sheng Zhang, and 1 others. 2023. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *CoRR*, abs/2311.16452.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2025. Introducing gpt-4.1 in the api: A new series of gpt models featuring major improvements on coding, instruction following, and long context—plus our first-ever nano model.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. 2024. Code generation with alphacodium: From prompt engineering to flow engineering. *CoRR*, abs/2401.08500.

Jon Saad-Falcon and 1 others. 2024. Archon: An architecture search framework for inference-time techniques. *CoRR*, abs/2409.15254.

Lloyd S. Shapley. 1953. A value for n-person games. In *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Trans. Assoc. Comput. Linguistics*, 10:539–554.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2024b. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 257–279. Association for Computational Linguistics.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. 2025. Agentdropout: Dynamic agent elimination for token-efficient and high-performance LLM-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Peng, and 1 others. 2023. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155.

Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. Haichart: Human and AI paired visualization system. *Proc. VLDB Endow.*, 17(11):3178–3191.

Yiheng Xu and 1 others. 2024. Lemur: Harmonizing natural language and code for language agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2369–2380. Association for Computational Linguistics.

Yilin Ye, Jianing Hao, Yihan Hou, Zhan Wang, Shishi Xiao, Yuyu Luo, and Wei Zeng. 2024. Generative AI for visualization: State of the art and future directions. *Vis. Informatics*, 8(1):43–66.

Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *CoRR*, abs/2406.07496.

Guibin Zhang, Yanwei Yue, Zhixun Li, and 1 others. 2025a. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, and 1 others. 2025b. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net.

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. 2025c. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*.

Jiayi Zhang, Chuang Zhao, Yihan Zhao, Zhaoyang Yu, Ming He, and Jianping Fan. 2024. Mobileexperts: A dynamic tool-enabled agent team in mobile devices. *CoRR*, abs/2407.03913.

Ruijia Zhang, Xinyan Zhao, Ruixiang Wang, Sigen Chen, Guibin Zhang, An Zhang, Kun Wang, and Qingsong Wen. 2025d. Safesieve: From heuristics to experience in progressive pruning for llm-based multi-agent communication. *CoRR*, abs/2508.11733.

Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Debug like a human: A large language model debugger via verifying runtime execution step by step. pages 851–870.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023. LLM as DBA. *CoRR*, abs/2308.05481.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, and 1 others. 2025. Mindstorms in natural language-based societies of mind. *Comput. Vis. Media*, 11(1):29–81.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Appendix

A Algorithm

Algorithm 1 Prune–Quantize–Fine-tune Pipeline

Input: initial graph $\mathcal{G}_{\text{base}}$, validation set $\mathcal{D}_{\text{val}}$, probe size m, budget B.
  Top-k size k, thresholds τ_p, τ_q.
  RRF weights $\mathbf{w}$, low-cost mapping π.
Output: optimized graph $\mathcal{G}^\star$.
Initialize $\mathcal{G} \leftarrow \mathcal{G}_{\text{base}}$.
Step 1: AgentPruner.
Set baseline $S_0 \leftarrow S(\mathcal{G})$.
while pruning budget remains **do**
  Rank nodes by probe signals; fuse by weighted RRF; get Top-k $\{v_1, v_2, \ldots, v_k\}$.
  for $v \in \{v_1, v_2, \ldots, v_k\}$ **do**
    $\mathcal{G}_p \leftarrow \text{Surgery}(\mathcal{G}, v)$; compute $S(\mathcal{G}_p)$.
    if $S(\mathcal{G}_p) \geq \tau_p \cdot S_0$ **then**
      $\mathcal{G} \leftarrow \mathcal{G}_p$; **break**
    end if
  end for
  Stop if no candidate is accepted.
end while
Step 2: AgentQuant.
Set baseline $S_0 \leftarrow S(\mathcal{G})$.
while quantization budget remains **do**
  Rank LLM nodes by probe replacement signals; fuse by weighted RRF; get Top-k $\{v_1, v_2, \ldots, v_k\}$.
  for $v \in \{v_1, v_2, \ldots, v_k\}$ **do**
    $\mathcal{G}_q \leftarrow \text{Replace}(\mathcal{G}, v, \pi(v))$; compute $S(\mathcal{G}_q)$.
    if $S(\mathcal{G}_q) \geq \tau_q \cdot S_0$ **then**
      $\mathcal{G} \leftarrow \mathcal{G}_q$; **break**
    end if
  end for
  Stop if no candidate is accepted.
end while
Step 3: AgentTuner.
Run MCTS on $\mathcal{G}$ with atomic edits; update $\mathcal{G}$.
return $\mathcal{G}^\star \leftarrow \arg\max_{\mathcal{H} \in \mathcal{K}} (S(\mathcal{H}), -C(\mathcal{H}))$.
  s.t. $C(\mathcal{G}^\star) \leq B$.

B Case Study

B.1 Workflow Overview and Legend Mapping

Figure 5 visualizes the pruning process on the redundant workflow graph generated on MATH (Hendrycks et al., 2021). To match the legend used in the figure, we map the implementation node IDs to the paper terms as follows: (1) `GenerateSolutionA` / `GenerateSolutionB` correspond to dual `AnswerGenerate` nodes. (2) `RefineWithCode` corresponds to the `CodeRefine` node. (3) `ScEnsembler` corresponds to the `ScEnsemble` node for aggregation. (4) `AnswerFormatter` formats the final output to satisfy benchmark constraints.

B.2 Pruning Rationale and Structural Simplification

The original workflow (Figure 5, left) contains multiple parallel reasoning paths: a program-execution path (`Programmer` → `RefineWithCode`), a detailed chain-of-thought path (`DetailedSolution`), and two additional solution generation paths (`GenerateSolutionA` and `GenerateSolutionB`). All candidate answers are routed into `ScEnsembler` for self-consistency selection before being formatted by `AnswerFormatter`. AgentSlimming identifies two main sources of redundancy: (1) The two `AnswerGenerate` branches are highly overlapping; keeping one branch provides most of the diversity gain. (2) The marginal utility of the `CodeRefine` step is limited for this workflow; its benefits do not justify the additional token cost.

C Experiment

C.1 Baselines

To ensure a fair and rigorous comparison that matches the scale of our optimized workflows, we configured the LLM-Debate baseline with **4 agents and 2 debate rounds**. This specific hyperparameter setting aligns the computational complexity and graph scale of the baseline with our method.

C.2 Hyperparameters

We report the key hyperparameters used across all experiments. The acceptance thresholds are set to $\tau_p = \tau_q = 0.95$, where higher values enforce more conservative selection and typically yield faster convergence, while lower values permit broader exploration. The top-k candidate pool size is set to k = 3 for both pruning and quantization evaluation. The RRF smoothing constant is set to $\kappa = \max(10, |V|)$. For the RRF importance weights, we set $w_{\text{degree}} = 1$, $w_{\text{betweenness}} = 1$, $w_{\text{shapley}} = 2$, and $w_{\Delta\text{cost}} = 1$, where the Shapley weight is assigned a higher value to emphasize cooperative contribution signals.

C.3 Nodes

Following AFlow, we treat an operator as a node in the DAG: each node corresponds to an executable operator (typically an LLM call) with fixed I/O constraints, including its own prompt, available resources/tools, and input–output schema. We provide complete per-node definitions and the default prompts for standard nodes in Section H.

C.4 Source Code

We provide the executable code for the initial workflow graphs of MATH, MBPP, and HotpotQA in Section I to ensure reproducibility.

D Stage-wise Contribution Details

In the AgentTuner stage, we adopt an offline workflow search procedure inspired by Monte Carlo Tree Search (MCTS). Each node in the search space corresponds to a complete workflow W, i.e., a full multi-agent system. At each iteration, the algorithm first selects a parent workflow from the historical candidate pool, and then invokes the LLM optimizer to generate a local modification based on both prior search experience and the operator set O. Such modifications may include prompt refinement, rewiring node connections, or recomposing operators, each yielding a new workflow candidate.

Different from classical UCT-style tree policies, our parent selection does not follow an explicit upper-confidence bound. Instead, we employ a soft mixed probability mechanism that interpolates between uniform exploration and score-based exploitation. Concretely, for n candidate parent workflows, the probability of selecting the i-th candidate is defined as:

$$P_{\text{mixed}}(i) = \lambda \cdot \frac{1}{n} + (1-\lambda) \cdot \frac{\exp\big(\alpha(s_i - s_{\max})\big)}{\sum_{j=1}^{n} \exp\big(\alpha(s_j - s_{\max})\big)} \tag{9}$$

where s_i denotes the average validation score of workflow i on the validation set $\mathcal{D}_{\text{val}}$, $s_{\max}$ is the maximum score among the current candidates, λ controls the exploration strength, and α controls the preference toward high-performing candidates. This design encourages broad exploration in the early stage while gradually biasing the search toward stronger workflows.

For each newly generated workflow, we evaluate its $(\text{score}, \text{cost})$ on $\mathcal{D}_{\text{val}}$. To reduce evaluation noise, the workflow can be executed multiple times and the average performance is recorded. The reward fed back to the search process is defined as the average validation score:

$$r(W) = \text{avgScore}(W; D_V) \tag{10}$$

The search is equipped with an early stopping rule: the procedure terminates when the top-k candidates remain unchanged for n consecutive rounds, or equivalently when the mean score of the top-k set no longer improves. The full hyperparameter configurations are as follows: λ = 0.3, α = 0.2, k = 3, stopping patience = 5, and the number of repeated evaluations = 1.

It is important to clarify the role of AgentTuner in our framework. The core contribution of this work lies in the structural optimization pipeline—namely, pruning and quantization—together with the RRF-based fusion of node-importance signals. AgentTuner is an optional offline enhancement module that can be applied when additional search budget is available, with the goal of further exploring the accuracy–cost Pareto frontier. It is not required for the validity of the proposed method, nor is it specifically designed to compensate for possible performance loss introduced by pruning or quantization.

The stage-wise results in Table 5 further support this interpretation. The pruning stage alone already delivers substantial cost reduction, and even improving accuracy on several datasets, indicating that structural simplification does not inherently rely on sacrificing task performance. The quantization stage yields additional cost savings, although it may incur moderate accuracy degradation on some tasks, which is consistent with the standard compression trade-off. Importantly, even without AgentTuner, the workflow obtained after pruning and quantization already achieves strong cost efficiency with acceptable performance, demonstrating that AgentTuner is not a mandatory recovery step for restoring model quality.

Moreover, AgentTuner does not always provide the best overall utility. On some datasets, it improves accuracy but also introduces a noticeable increase in inference cost. This observation further confirms that AgentTuner should be viewed as an optional module rather than a necessary corrective component of the system. In practice, it is most beneficial in scenarios where high accuracy is prioritized over cost, where compression causes a non-trivial accuracy drop and partial recovery is desired, or where sufficient budget is available to search for solutions closer to the Pareto-optimal frontier.

Method	MATH		AIME		GSM8K		HotpotQA	
	Score	Cost ($/prob.)	Score	Cost ($/prob.)	Score	Cost ($/prob.)	Score	Cost ($/prob.)
Original	73.1	1.03e-2	66.4	1.55e-2	96.6	4.34e-3	77.0	7.65e-3
After pruning	75.6	3.30e-3	67.1	1.19e-2	95.8	2.18e-3	76.7	5.55e-3
After quantization	74.8	2.54e-3	67.1	1.19e-2	93.9	5.70e-4	72.8	2.90e-3
After tuning	75.6	1.04e-2	69.6	1.28e-1	95.1	6.10e-4	71.9	4.20e-3
Method	**MBPP**		**DROP**		**LiveCode**		**MusiqueAns**	
	Score	Cost ($/prob.)	Score	Cost ($/prob.)	Score	Cost ($/prob.)	Score	Cost ($/prob.)
Original	69.8	2.57e-3	77.9	1.55e-3	60.2	1.09e-2	85.3	1.07e-2
After pruning	79.7	1.63e-3	77.7	6.40e-4	59.8	4.55e-3	86.4	3.51e-3
After quantization	79.2	7.10e-4	78.4	4.50e-4	61.4	3.20e-3	85.3	9.90e-4
After tuning	80.2	7.10e-4	83.3	9.50e-4	61.7	2.59e-3	85.8	9.90e-4

Table 5: **Stage-wise contribution analysis of pruning, quantization, and AgentTuner.** We report the score and average inference cost per problem at different stages of the optimization pipeline across eight representative benchmarks. The results show that pruning alone already yields substantial cost reduction and can even improve task performance on several datasets. Quantization further reduces cost, while sometimes introducing a moderate accuracy drop, reflecting the standard compression trade-off. AgentTuner serves as an optional offline enhancement step that may further improve performance on some tasks, but it is not required for obtaining strong cost efficiency, nor does it always provide the best overall cost–accuracy trade-off.

E Break-even Analysis of Optimization Cost

To quantify the trade-off between the one-time optimization overhead and the recurring inference-time savings, we provide a time-to-break-even analysis. Let C_{optimize} denote the total cost incurred during the structural optimization/search stage. Let C_{base} and C_{ours} denote the average API cost per inference before optimization (the baseline method) and after optimization (our method), respectively. Then, the per-inference cost saving is defined as:

$$\Delta C = C_{\text{base}} - C_{\text{ours}} \tag{11}$$

Accordingly, the break-even point is defined as the number of future executions required to amortize the one-time optimization cost:

$$N_{\text{break-even}} = \frac{C_{\text{optimize}}}{\Delta C} \tag{12}$$

Here, C_{optimize} is paid only once during the optimization stage, whereas ΔC yields recurring savings in every subsequent deployment. Therefore, as the optimized workflow is executed repeatedly, the upfront search cost is gradually amortized over time.

Table 6 reports the benchmark-wise C_{optimize} and the resulting $N_{\text{break-even}}$ values relative to AFlow, providing a more complete view of the trade-off between search overhead and inference-time savings.

Datasets	Total optimization cost ($)	Break-even inference calls
AIME	43.07	13459
MATH	14.04	2742
DROP	2.26	3767
MBPP	2.26	1222
GSM8K	5.70	1652
HotpotQA	18.57	8889
LiveCode	26.52	2873
MusiqueAns	25.85	10508

Table 6: **Break-even analysis of optimization cost.** We report the total one-time optimization cost C_{optimize} and the corresponding number of future executions required to amortize this cost relative to AFlow.

Probe Size	HotpotQA	
	Acc.	Cost ($/prob.)
10	75.6	5.89e-3
50	77.0	5.55e-3
100	75.3	3.79e-3

Table 7: **Sensitivity analysis of probe dataset size on HotpotQA.** We vary the probe size used for node-importance evaluation and report the resulting task accuracy and average inference cost per problem.

Method	MATH		HotpotQA	
	Acc.	Cost	Acc.	Cost
Baseline (ADAS)	69.0	1.15e-2	81.6	5.89e-3
Ours	70.4	9.67e-3	79.4	2.75e-3

Table 8: **Experiments on ADAS-based workflows.** We apply AgentSlimming to workflows derived from ADAS without changing the core optimization algorithm or hyperparameter settings.

Method	AIME		HotpotQA		MBPP		GSM8K	
	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)
Baseline (AFlow)	87.1	1.19e-2	81.6	5.89e-3	73.3	2.84e-3	96.6	3.04e-3
Ours	87.9	6.59e-3	79.4	2.76e-3	75.6	1.06e-3	97.9	5.57e-4

Table 9: **Cross-architecture validation on the Qwen3 series.** We replace the high-/low-cost nodes with `qwen3-235b-a22b-instruct-2507` and `qwen3-30b-a3b-instruct-2507`, respectively, while keeping all other algorithms and hyperparameters unchanged. Across AIME, HotpotQA, MBPP, and GSM8K, AgentSlimming consistently achieves substantial cost reductions while maintaining competitive overall performance.

Method	MATH		GSM8K	
	Acc.	Cost ($/prob.)	Acc.	Cost ($/prob.)
AFlow	74.8	1.20e-2	95.5	4.38e-3
Ours	73.9	6.88e-3	95.5	9.30e-4
Cross-dataset results	77.3	6.63e-3	96.6	2.63e-3

Table 10: **Cross-dataset validation between GSM8K and MATH.** We optimize the workflow structure on one dataset and directly evaluate the resulting structure on the other, without changing the optimization algorithm or hyperparameter settings. The transferred structures remain competitive in both accuracy and cost efficiency, suggesting that AgentSlimming exhibits non-trivial cross-dataset generalization across same-type reasoning tasks.

F Probe Dataset Size Sensitivity

To evaluate the sensitivity of our method to the probe dataset size, we conduct additional experiments on HotpotQA with probe sizes of 10, 50, and 100. The results are summarized in Table 7. We observe that even a moderate probe size of 50 is sufficient to produce a stable node-importance ranking and achieves the best accuracy among the three settings. Further increasing the probe size to 100 reduces inference cost, but also leads to a drop in accuracy. Empirically, a probe size in the range of 30–60 already provides strong performance, and the importance scores stabilize with as few as 50 probe samples. This suggests that our optimization process does not require a large probe dataset to obtain consistent pruning decisions, which further improves the practicality and efficiency of the overall framework.

G Additional Generalization Studies

G.1 Experiments on the Qwen3 Series

To evaluate whether our method generalizes across model families, we further conduct cross-architecture validation on the Qwen3 series. Specifically, we replace the high-/low-cost nodes with `qwen3-235b-a22b-instruct-2507` and `qwen3-30b-a3b-instruct-2507`, respectively, while keeping all other algorithms and hyperparameters unchanged. We evaluate on four datasets: AIME, HotpotQA, MBPP, and GSM8K, which cover both high- and low-difficulty settings as well as diverse task types.

The results are summarized in Table 9. We observe that AgentSlimming still delivers substantial cost reductions under this cross-architecture setting, while preserving competitive overall accuracy. In particular, our method improves accuracy on AIME, MBPP, and GSM8K, while incurring only a moderate drop on HotpotQA. These results suggest that the effectiveness of AgentSlimming is not tied to a specific model family, and that its cost-saving benefits transfer well to the Qwen3 series.

G.2 Cross-Dataset Validation between GSM8K and MATH

To further address concerns about generalization to unseen data, we conduct additional cross-dataset validation between GSM8K and MATH. Specifically, we optimize the workflow structure on one dataset and then directly evaluate the resulting structure on the other dataset. No additional structural search is performed on the target dataset, and all optimization algorithms and hyperparameter settings are kept unchanged.

Table 10 presents the cross-dataset results. These results suggest that the optimized structures discovered by AgentSlimming are not overly specialized to a single dataset. Instead, they transfer reasonably well across same-type reasoning datasets, indicating non-trivial cross-dataset generalization ability.

G.3 Experiments on ADAS-Based Workflows

To further verify that AgentSlimming does not rely on AFlow-specific implementations, we conduct an additional cross-framework transfer study by applying the same slimming pipeline to ADAS, an MAS framework that differs from AFlow in both its optimization mechanism and overall system organization, under the same tasks and evaluation protocol. We keep the core optimization algorithm and hyperparameter settings unchanged, and evaluate the resulting compressed workflows on MATH and HotpotQA.

The results are summarized in Table 8. We observe that AgentSlimming remains effective on ADAS-based workflows, achieving substantial cost reduction while keeping performance degradation within an acceptable range, consistent with the trend observed on AFlow. These results support the robustness and transferability of AgentSlimming beyond a single workflow family.

More broadly, AgentSlimming only assumes that the underlying multi-agent workflow can be modeled as a DAG. Under this abstraction, we estimate an importance score for each agent node and iteratively perform removal or low-cost replacement, with a baseline-anchored acceptance rule to prevent performance collapse. Therefore, AgentSlimming can be used as a plug-and-play module for a broad range of graph-structured MAS, as long as the workflow can be abstracted into nodes and dependencies and the system provides a stable task-level evaluation signal. We also note that systems that cannot be reasonably expressed as a DAG, or that lack a stable evaluation signal, may require additional graph transformation or evaluation design.

H Nodes

H.1 Definitions of Standard Nodes

Node name	Role	Functionality	Applicable datasets
AnswerGenerate Node	Answer Generator	Directly generates an answer to the input question using an LLM. It is the standard node for direct question answering.	General QA, Reasoning (e.g., GSM8K, MATH, HotpotQA)
Programmer Node	Program Executor	Generates Python code (PoT/PAL style) to solve the problem and executes it in a local sandbox. It includes a retry mechanism for execution failures.	Math & Logic tasks requiring calculation (e.g., MATH, GSM8K)
CustomCode-Generate Node	Code Generator	Focuses solely on generating executable Python code (e.g., a solve() function) based on the problem, without executing the code immediately.	Code Generation (e.g., LiveCode, MBPP)
Test Node	Self-Corrector	Executes generated code and iteratively refines/fixes it based on execution feedback (runtime errors or incorrect outputs).	Complex Reasoning & Coding (e.g., LiveCode, MBPP)
ScEnsemble Node	Ensemble Selector	Aggregates multiple candidate solutions from upstream nodes and selects the best one using Self-Consistency or majority voting logic.	General Reasoning (e.g., GSM8K, MATH)
CodeRefine Node	Code Refiner	Refines and improves an existing draft solution by leveraging programmatic reasoning or code-based feedback before final aggregation/formatting.	Math & Multi-step Reasoning with program-aided refinement (e.g., GSM8K, MATH, AIME)
Custom Node	Custom Operator	Executes a user-defined prompt or custom logic, providing flexibility for specific sub-tasks not covered by standard nodes.	Any / Agnostic

H.2 Default Prompts for Standard Nodes

Prompt of Standard Nodes

```
PROMPT_ANSWERGENERATOR = """
    Think step by step and solve the problem.
    1. In the "thought" field, explain your thinking process in detail
        .
    2. In the "answer" field, provide the final answer concisely and
        clearly. The answer should be a direct response to the
        question, without including explanations or reasoning.

    Your task: {input}
"""


PROMPT_CUSTOMCODEGENERATE = """
    You are a professional Python programmer. Your task is to write
        complete, self-contained code based on a given problem and
        output the answer. The code should include all necessary
        imports and dependencies, and be ready to run without
        additional setup or environment configuration.

    Problem: {problem}
    Resources: {input}

    Your code should:
    1. Implement the calculation steps described in the problem.
    2. Define a function named `{function_name}` that performs the
        calculation and returns the result.
    3. Return only runnable Python code without explanations.
"""


PROMPT_SCENSEMBLE = """
    Given the problem described as follows: {problem}
    Several candidate solutions have been generated to address the
        given problem. They are as follows:
    {solutions}

    Your task is to act as an expert evaluator. Carefully evaluate
        these solutions and identify the definitive answer that
        appears most frequently across them (Majority Consensus).
    Analyze the solutions to determine the most consistent and
        reliable outcome.
"""


PROMPT_PROGRAMMER = """
    You are a professional Python programmer. Your task is to write
        complete, self-contained code based on a given problem and
        output the answer. The code should include all necessary
        imports and dependencies, and be ready to run without
        additional setup or environment configuration.

    Problem: {problem}
    Resources: {input}

    Your code should:
    1. Implement the calculation steps described in the problem.
    2. Define a function named `solve` that performs the calculation
        and returns the result. The `solve` function should not
        require any input parameters; instead, it should obtain all
        necessary inputs from within the function or from globally
        defined variables.
    3. `solve` function return the final calculation result.
```

```
    Please ensure your code is efficient, well-commented, and follows
        Python best practices. The output should be limited to basic
        data types such as strings, integers, and floats. It is
        prohibited to transmit images or other file formats. The code
        output is intended for a text-based language model.
"""

PROMPT_TEST = """
    Given a problem and a python code solution which failed to pass
        test or execute, you need to analyze the reason for the
        failure and propose a better code solution.

    Problem: {problem}

    Failure details:
    {error_info}

    Please provide a "reflection" field explaining the failed test
        cases and code solution, and a "solution" field containing a
        better code solution without any additional text or test cases
        .
"""


PROMPT_ANSWERFORMAT = """
    You are an answer formatter for the {dataset_name} dataset.

    Task Context:
    {task_context}

    Format Requirements:
    {format_requirements}

    Original Answer:
    {original_answer}

    Return only the final formatted answer according to the
        requirements.
"""
```

I Examples of Initial Workflow Graphs

I.1 Minimal initial workflow graph on MATH dataset

```python
from src.core.edge import Edge
from src.core.graphflow import GraphFlow
from src.core.nodes.answer_format_node import AnswerFormatNode
from src.core.nodes.custom_node import CustomNode
from src.core.nodes.input_node import InputNode
from src.core.workflow import BaseWorkflow
from . import prompt as prompt_custom

class Workflow(BaseWorkflow):
    def _build_graph(self) -> GraphFlow:
        Input = InputNode(
            node_id="Input",
            node_llm_config=self.llm_config,
            description="Graph input entry.",
        )
        Reasoner = CustomNode(
            node_id="Reasoner",
            node_prompt=prompt_custom.PROMPT_REASONING,
            node_llm_config=self.llm_config,
            description="Reason over the problem and produce a concise
                answer.",
```

```
        )
        AnswerFormatter = AnswerFormatNode(
            node_id="AnswerFormatter",
            dataset_name=self.dataset,
            node_llm_config=self.llm_config,
            description="Format the final answer for the target
                dataset.",
        )
        return GraphFlow(
            nodes=[Input, Reasoner, AnswerFormatter],
            edges=[
                Edge(source="Input", target="Reasoner"),
                Edge(source="Reasoner", target="AnswerFormatter"),
            ],
            entry_node_ids=["Input"],
            final_node_id="AnswerFormatter",
            description="Minimal MATH round_1 workflow.",
        )
```

Prompt of a *Reasoner* Custom Node

```
PROMPT_REASONING = (
    "Solve the problem carefully. Identify the key facts, reason step
        by step, and return a concise answer."
)
```

I.2 Minimal initial workflow graph on MBPP dataset

```
from src.core.edge import Edge
from src.core.graphflow import GraphFlow
from src.core.nodes.answer_format_node import AnswerFormatNode
from src.core.nodes.custom_code_generate_node import
    CustomCodeGenerateNode
from src.core.nodes.input_node import InputNode
from src.core.workflow import BaseWorkflow

class Workflow(BaseWorkflow):
    def _build_graph(self) -> GraphFlow:
        Input = InputNode(
            node_id="Input",
            node_llm_config=self.llm_config,
            description="Graph input entry.",
        )
        CustomCodeGenerator = CustomCodeGenerateNode(
            node_id="CustomCodeGenerator",
            node_llm_config=self.llm_config,
            description="Generates Python code for code-generation
                benchmarks and preserves the entry point.",
        )
        AnswerFormatter = AnswerFormatNode(
            node_id="AnswerFormatter",
            dataset_name=self.dataset,
            node_llm_config=self.llm_config,
            description="Format the final answer for the target
                dataset.",
        )
        return GraphFlow(
            nodes=[Input, CustomCodeGenerator, AnswerFormatter],
            edges=[
                Edge(source="Input", target="CustomCodeGenerator"),
                Edge(source="CustomCodeGenerator", target="
                    AnswerFormatter"),
            ],
```

```
            entry_node_ids=["Input"],
            final_node_id="AnswerFormatter",
            description="Minimal MBPP round_1 workflow.",
        )
```

I.3 Minimal initial workflow graph on HotpotQA dataset

```
from src.core.edge import Edge
from src.core.graphflow import GraphFlow
from src.core.nodes.answer_format_node import AnswerFormatNode
from src.core.nodes.answer_generate_node import AnswerGenerateNode
from src.core.nodes.input_node import InputNode
from src.core.workflow import BaseWorkflow

class Workflow(BaseWorkflow):
    def _build_graph(self) -> GraphFlow:
        Input = InputNode(
            node_id="Input",
            node_llm_config=self.llm_config,
            description="Graph input entry.",
        )
        AnswerGenerator = AnswerGenerateNode(
            node_id="AnswerGenerator",
            node_llm_config=self.llm_config,
            description="Built-in answer generation node for direct
                step-by-step solving.",
        )
        AnswerFormatter = AnswerFormatNode(
            node_id="AnswerFormatter",
            dataset_name=self.dataset,
            node_llm_config=self.llm_config,
            description="Format the final answer for the target
                dataset.",
        )
        return GraphFlow(
            nodes=[Input, AnswerGenerator, AnswerFormatter],
            edges=[
                Edge(source="Input", target="AnswerGenerator"),
                Edge(source="AnswerGenerator", target="AnswerFormatter
                    "),
            ],
            entry_node_ids=["Input"],
            final_node_id="AnswerFormatter",
            description="Minimal HotpotQA round_1 workflow.",
        )
```