Rethinking the Value of Homogeneous Multi-Agent Workflows: A Strong Single-Agent Baseline

Anonymous ACL submission

Abstract

Recent advances in LLM-based multi-agent systems (MAS) show that workflows composed of multiple LLM agents with distinct roles, tools, and communication patterns can outperform single-LLM baselines on complex tasks. In this paper, we focus on a common class of *homogeneous* workflows where agents share the same base LLM and differ mainly in prompts, tools, and positions in the workflow. Our formulation focuses on serialized execution with shared visible history, rather than private-state or asynchronous coordination. This raises the question of whether such workflows can be simulated by a single agent through multi-turn conversations. Our results on seven benchmarks spanning coding, mathematics, general question answering, domain-specific reasoning, and real-world planning show that single-agent execution can achieve comparable performance to these homogeneous workflows, with cost reductions from KV-cache reuse for fixed workflows. We propose OneFlow, a cost-aware workflow search algorithm that automatically tailors workflows for single-agent execution. Its larger cost reductions come from combining streamlined workflow search with single-agent execution, while maintaining comparable accuracy. These results position the single-LLM implementation of homogeneous workflows as a strong baseline in this setting. We also note that single-LLM methods cannot capture heterogeneous workflows due to the lack of KV cache sharing across different LLMs, highlighting future opportunities for developing heterogeneous multi-agent systems.

1 Introduction

Recent advances in large language models (LLMs) (Brown et al., 2020; Ouyang et al., 2022) have sparked significant interest in multi-agent systems (MAS) (Li et al., 2023; Hong et al., 2024), where multiple LLM agents collaborate through predefined workflows to tackle complex tasks. These systems typically consist of specialized LLM agents, each defined by distinct system prompts and tools, that communicate according to specific patterns and are often reported to perform competitively on complex tasks (Zhuge et al., 2024; Liu et al., 2024; Zhang et al., 2025e). Current research has reported promising results for such multi-agent workflows across diverse domains, from mathematical reasoning to code generation and tool usage (Hu et al., 2025; Zhang et al., 2025b; Wang et al., 2025c).

However, we focus on a practically important subset of MAS: **homogeneous, collaborative workflows** (Ye et al., 2025a; Zhang et al., 2025a), where agents share the same base LLM and differ mainly in system prompts, tools, and positions in the workflow. This setting is common in automatic workflow-design systems, where the main design variables are often prompts, tools, and routing rather than model identity. Our scope is serialized workflow execution with shared visible context, and does not cover settings centered on private per-agent state or asynchronous interaction. This scoped setting raises a compelling question: if agents share the same underlying model and collaborate toward one task, can a single agent simulate the workflow effectively through multi-turn conversations? (Li, 2026) Task decomposition is a well-established strategy for solving complex problems (Wei et al., 2022; Zhuge et al., 2025), and directly motivates the design of multi-agent systems. Because homogeneous agents possess identical reasoning capabilities and differ only in their specialized instructions, a single agent may be capable of role-playing these agents sequentially, thereby exploiting the workflow's task decomposition (Li, 2026). Moreover, a single agent can reuse a shared KV cache across agent interactions (Xiao et al., 2024), retaining context without additional prefill cost and potentially offering efficiency gains

and greater consistency than maintaining separate model instances for each agent.

To investigate this hypothesis, we conduct experiments across seven benchmarks spanning coding, mathematics, general question answering, domain-specific reasoning, and real-world planning and tool use. Our results reveal that *a single agent using multi-turn conversations with KV cache can often simulate tailored homogeneous workflows with performance comparable to traditional homogeneous multi-agent setups, while reducing cost*. This finding suggests that, under the assumptions in Section 3.1, multiple separate agent instances are not always required for effective tailored reasoning in our scoped setting. Building on this insight and recent work on automatic workflow design (Zhang et al., 2025e,b; Nie et al., 2025; Hu et al., 2025), we introduce **OneFlow**, an algorithm for automatically designing tailored workflows that optimize both performance and computational efficiency for single-agent execution.

A natural caveat is that single-LLM execution cannot capture *heterogeneous* workflows, since KV cache cannot be shared across different base models. As a limited pilot study, we use AFlow (Zhang et al., 2025e) to automatically design one heterogeneous workflow with different base LLMs (Ye et al., 2025a; Zhang et al., 2025a; Wang et al., 2025b). The results are mixed: the heterogeneous workflow is strongest on some tasks, but does not consistently dominate homogeneous alternatives and often incurs higher cost. Our contributions can be summarized as follows:

- Empirically validate that single-agent execution can effectively simulate **homogeneous** multi-agent workflows with comparable performance on collaborative tasks.
- Propose **OneFlow**, a cost-aware automatic workflow design algorithm that explicitly optimizes the performance–cost trade-off, producing streamlined workflows well-suited for single-agent execution.
- Include a limited heterogeneous pilot study, highlighting mixed cost-performance trade-offs and the need for more systematic evaluation of heterogeneous workflow design.

2 Preliminary

LLM-based Multi-Agent Workflows. Multi-agent workflows represent a paradigm where multiple LLM agents collaborate through structured communication patterns to solve complex tasks. To illustrate this concept, consider the session-based query recommendation example shown in Figure 1. Given a customer's shopping history and current query, the task requires understanding session context, analyzing product relationships, and generating relevant recommendations, a multi-faceted problem that benefits from specialized processing stages, as demonstrated in the middle panel of Figure 1. Importantly, these workflows are implemented as executable Python code (right panel of Figure 1) that specifies both the agents and their interaction logic. This code-based representation enables complex control flows including sequential execution, conditional branching, and iterative refinement, etc.

Formal Definition. We now provide a characterization of LLM multi-agent workflows. Let $\mathcal{M}$ denote the set of available base LLMs. An LLM-based multi-agent workflow W is defined as a directed graph $G = (N, E)$ where:

- $N = \{n_1, n_2, ..., n_{|N|}\}$ represents the set of LLM agents. Each agent n_i is parameterized as $n_i = (b_i, p_i, \tau_i)$, where: $b_i \in \mathcal{M}$ is the base LLM (e.g., Claude 4.5 Haiku, Gemini 3 Flash), p_i is the system prompt that defines the agent's role and capabilities, τ_i is the available tool set (e.g., sandboxed Python interpreter, web search). Agents are typically specialized for specific subtasks, with common roles including LLM reviewers and LLM ensemblers, etc.
- $E \subseteq N \times N$ encodes the inter-agent communication structure and control flow. Each edge may include routing conditions and message transformations implemented in Python, enabling sophisticated orchestration patterns such as sequential processing, conditional branching, and iterative loops.

Homogeneous vs. Heterogeneous Workflows. A critical distinction in multi-agent workflows concerns the diversity of underlying base models. Let $\mathcal{B}(W) = \{b_i \mid n_i \in N\}$ denote the set of base LLMs used by workflow W. A workflow is *homogeneous* when $|\mathcal{B}(W)| = 1$, i.e., all agents share the same base LLM and differ only in system prompts, tools, and positions in the workflow; it is *heterogeneous* when $|\mathcal{B}(W)| > 1$, allowing different agents to use different base LLMs.

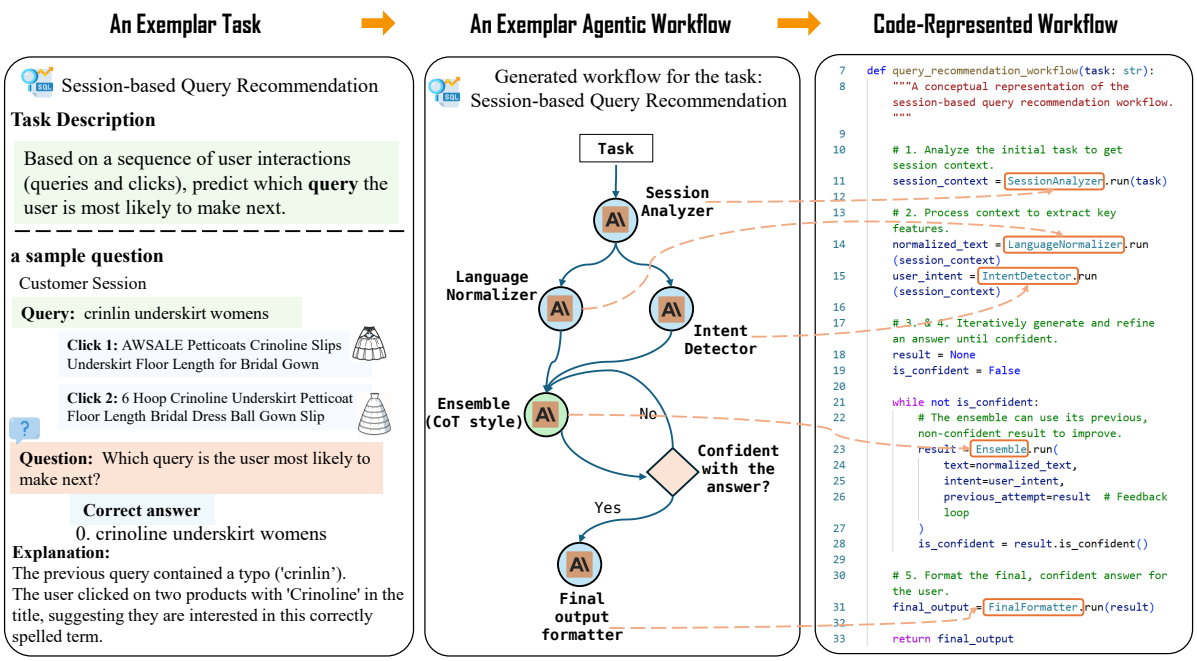

Figure 1: Sample question–answer pair from a session-based query recommendation task (left), an exemplar agentic workflow to solve it (middle), and its code representation (right). The workflow demonstrates how multiple LLM agents can collaborate to process complex shopping queries through sequential and conditional execution patterns.

The distinction between heterogeneous and homogeneous workflows is fundamental to our analysis, as it determines whether a workflow can be efficiently simulated by a single agent instance through multi-turn conversations with shared KV cache. In transformer-based LLMs, the KV cache stores computed key-value matrices from attention layers for previously processed tokens, avoiding redundant recomputation during autoregressive generation. In homogeneous workflows, agents share the same base LLM, enabling KV cache reuse across agent turns: shared context (task descriptions, intermediate results) need only be encoded once, reducing prefill cost and improving consistency compared to separate model instances. Heterogeneous workflows offer additional flexibility, but they also introduce model selection into the design space, which we treat as a separate direction.

3 Methodology

3.1 Single-agent execution can match multi-agent frameworks.

We formalize when a single agent can implement a homogeneous multi-agent workflow without loss of expressivity. Recall the formalization in the Preliminary section: a workflow W is a directed graph $G = (N, E)$ with agents $n_i = (b_i, p_i, \tau_i)$ and routing logic on edges. Let W be homogeneous with $|\mathcal{B}(W)| = 1$ and base LLM b. Executing W on input x produces a transcript

$$H_T = (h_0, m_1, r_1, h_1, \ldots, m_T, r_T, h_T),$$

where h_0 is the initial context derived from input x. At each step t, the workflow: (1) selects agent index i_t according to a routing policy $\pi_W(i_t \mid h_{t-1})$ induced by E; (2) queries base model b with system prompt p_{i_t} and context h_{t-1} to obtain model message m_t; (3) if m_t requests a tool from τ_{i_t}, executes it to obtain result r_t (otherwise $r_t = \emptyset$); and (4) updates the history $h_t = \mathsf{append}(h_{t-1}, m_t, r_t)$.

Consider a single-LLM simulator that maintains one conversation state and, at each step t, sets the active system message to p_{i_t}, carries forward context h_{t-1}, and decodes from base model b with identical decoding parameters.

Proposition 1 (Simulation of homogeneous workflows). Suppose (i) tool side-effects are deterministic given inputs, (ii) the routing policy π_W depends only on the visible history h_{t-1}, and (iii) decoding uses deterministic rules (e.g., greedy) or shared randomness. Then the single-LLM simulator induces the same distribution over transcripts as executing W with separate agent instances:

$$H_T^{\text{single}} \stackrel{d}{=} H_T^{\text{multi}}.$$

Proof sketch. Both procedures query the same conditional distribution $b(\cdot \mid p_{i_t}, h_{t-1})$ at the same sequence of states; induction on t yields equality in distribution.

This equivalence is scoped: it can break when a workflow relies on private per-agent memory, routing decisions not recoverable from the shared visible history, stochastic or stateful tool side effects, asynchronous execution, or parallel writes to shared state.

Cost with KV cache. Let L_t denote the tokenized length of the prefix visible at step t, ΔL_t the number of new tokens appended between steps $t-1$ and t, and G_t the number of tokens generated at step t. With separate agent instances (no cache sharing), overlapping prefixes must be re-encoded at each step, yielding cost

$$C_{\text{multi}} \propto \sum_{t=1}^{T} (L_t + G_t).$$

The single-LLM simulator reuses the KV cache across steps, so

$$C_{\text{single}} \propto \sum_{t=1}^{T} (\Delta L_t + G_t) \;\leq\; C_{\text{multi}},$$

with equality only when agent contexts are disjoint. Thus, for homogeneous workflows with substantial context overlap, single-LLM execution is asymptotically no worse and often cheaper, while preserving behavior under the conditions above.

3.2 Single agent implementation of multi-agent workflow.

We provide a concrete single-LLM simulator for any homogeneous workflow $W = (N, E)$ with base LLM b. Let $n_i = (b, p_i, \tau_i)$. The simulator maintains a single chat history h_t and follows the routing policy encoded by E.

1. **Initialization.** Set $h_0 = \texttt{wrap}(x)$ with task x and global instructions. Insert role delimiters to isolate subsequent agent turns.

2. **Agent step t.** Using the routing logic in E, select agent index i_t based on h_{t-1}. Set the active system message to p_{i_t}, while keeping all prior messages and tool results in visible context h_{t-1}. Then, query b to obtain $m_t \sim b(\cdot \mid p_{i_t}, h_{t-1})$ with fixed decoding parameters.

3. **Tool execution.** If m_t requests a tool $u \in \tau_{i_t}$, execute u to obtain result r_t; otherwise set $r_t = \emptyset$.

4. **State update.** Set $h_t = \texttt{append}(h_{t-1}, m_t, r_t)$ and advance according to edge conditions in E. Repeat until termination.

Because every step calls the same base model b, the simulator reuses the KV cache across steps (see Preliminary), so the prefill cost scales with incremental growth ΔL_t rather than the full prefix L_t. To control context growth and mitigate interference, one may insert compaction operators (e.g., deterministic summarization) that map a window of turns $h_{t-k:t} \mapsto s_t$ where s_t is the summarized representation, while preserving routing decisions, which leaves Proposition 1 applicable.

3.3 OneFlow: Automatic Workflow Design for Single-Agent Execution

Sections 3.1–3.2 established that a single agent can efficiently simulate any homogeneous workflow with KV cache reuse. Building on this insight, OneFlow automatically searches for multi-agent workflows that are both effective and cost-efficient.

For task $\mathcal{T}$, we seek an optimal workflow W^* that achieves high performance at low cost:

$$W^* = \arg\max_{W \in \mathcal{W}} [\alpha \cdot P(W, \mathcal{T}) - \beta \cdot C(W, \mathcal{T})] \quad (1)$$

where $P(W, \mathcal{T})$ is performance (e.g., accuracy), $C(W, \mathcal{T})$ is inference cost, and α, β balance the two objectives.

We search within *homogeneous* workflows, ensuring any discovered workflow can be executed by a single agent with KV cache reuse (Section 3.1). We adopt the MCTS framework from AFlow (Zhang et al., 2025e), but with a key difference: *our optimization objective explicitly includes cost-efficiency*, not just task performance. This produces more streamlined workflows that work well with single-agent execution. The full procedure is summarized in Algorithm 1 and illustrated in Figure 2.

Initialization. We begin with a simple input-output workflow W_{IO} as the root node, evaluating it on a validation set D_v (20% of samples) to measure both performance and cost. The root node also stores failure cases to guide subsequent expansions.

MCTS Process. We follow a standard four-stage MCTS approach: (1) *Selection*: choose a workflow W from the tree based on a performance-weighted distribution; (2) *Expansion*: generate a

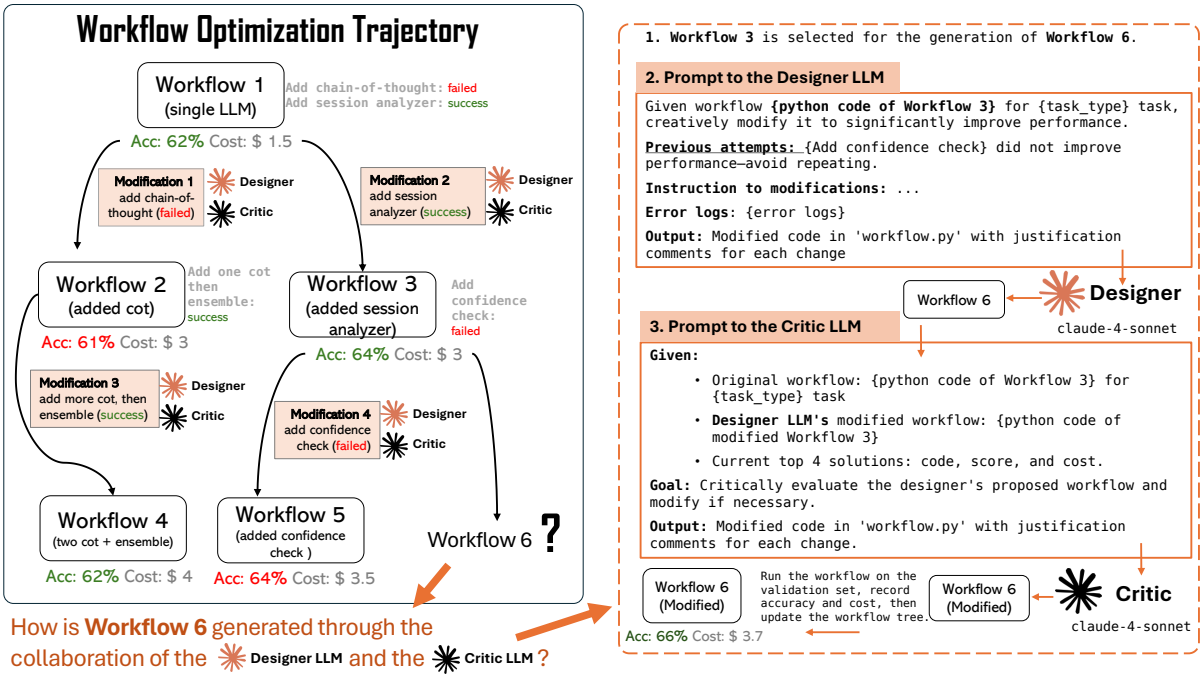

Figure 2: Overview of OneFlow. The framework uses MCTS with dual meta-LLMs to search for workflows optimized for single-agent execution. The left panel shows the iterative search process; the right panel illustrates how a new workflow is generated through the Creative Designer and Critical Reviewer.

Algorithm 1 OneFlow: Cost-Aware Workflow Search

Require: Task $\mathcal{T}$, validation set D_v, iterations K, weights α, β
Ensure: Optimized workflow W^*
  $W_{\text{IO}} \leftarrow$ simple input-output workflow; Eval on D_v; init tree
  for $k = 1$ to K **do**
    $\mathcal{C} \leftarrow \{W_{\text{IO}}\} \cup$ Top-$(n-1)$ by score $S(W) = \alpha P(W) - \beta C(W)$
    $W \leftarrow \text{Select}(\mathcal{C})$ ▷ Selection
    $W_d \leftarrow \text{Designer}(W, \text{errors}(W))$ ▷ Propose improvements
    $W_{\text{new}} \leftarrow \text{Reviewer}(W_d)$ ▷ Refine
    Eval(W_{new} on D_v); attach to tree; backpropagate
  end for
  return $\arg\max_W [\alpha P(W, \mathcal{T}) - \beta C(W, \mathcal{T})]$

new workflow W_{new} using dual meta-LLMs (a Creative Designer proposes performance-focused improvements, a Critical Reviewer refines them for cost-efficiency); (3) *Evaluation*: test W_{new} on D_v to obtain performance and cost metrics; (4) *Back-propagation*: update the tree and record modifications to avoid redundant designs. This iterates for 20 rounds. Implementation details are in Appendix A.4 and Appendix A.5.

Summary. OneFlow operates in two stages: (1) *Workflow Search* using MCTS to discover an optimized homogeneous workflow W^*; (2) *Single-Agent Execution* via the simulator in Section 3.2. In experiments, **OneFlow** denotes multi-agent execution of the searched workflow, while **OneFlow (single-agent execution)** denotes single-LLM simulation with KV cache reuse.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our approach across a diverse set of benchmarks spanning multiple domains to assess generalization capabilities. Our evaluation suite includes: (*i*) *code generation tasks*: MBPP and HumanEval; (*ii*) *mathematical reasoning*: GSM8K and MATH; (*iii*) *question answering*: HotpotQA and DROP; (*iv*) *domain-specific reasoning*: Shopping-MMLU; and (*v*) *real-world planning and tool use*: TravelPlanner.

Evaluation Metrics. We assess both task performance and computational efficiency. For code generation, we report `pass@1` accuracy; for general question answering, we use F1 score; for mathematical tasks, we report solve rate (%); for Shopping-MMLU, we use accuracy. For TravelPlanner, we use task success rate (%). Unless otherwise specified, cost tables report deployment-time executor inference cost for running a finalized workflow, measured as USD token expenditure over input and output tokens. Workflow search/design cost is a

Table 1: Main results on public benchmarks with GPT-4o-mini as executor. Values are mean ± std over three runs, in percentage (0–100). Code tasks report pass@1; QA tasks report F1; solve rate (%) for math. Best per column in **bold**; runner-up underlined.

	CODE		MATH		QA	
Method	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
Manual baselines						
IO	89.1 ± 0.4	72.6 ± 0.3	87.0 ± 0.1	51.3 ± 0.6	71.1 ± 0.5	66.2 ± 0.6
CoT (Wei et al., 2022)	90.3 ± 1.2	73.3 ± 0.0	87.1 ± 0.2	50.9 ± 0.3	71.2 ± 1.0	78.9 ± 0.4
CoT SC (5-shot)	89.8 ± 0.9	71.9 ± 1.0	92.6 ± 0.8	37.7 ± 1.2	67.3 ± 0.3	79.4 ± 0.1
MultiPersona (Wang et al., 2024)	89.1 ± 0.2	73.3 ± 0.3	87.1 ± 0.1	50.9 ± 0.3	71.2 ± 1.0	78.9 ± 0.4
Automatically designed multi-agent frameworks						
AFlow (Zhang et al., 2025e)	90.1 ± 0.0	78.8 ± 0.7	**93.6 ± 0.5**	**55.6 ± 0.3**	72.1 ± 0.2	**83.1 ± 0.3**
OneFlow	91.6 ± 0.8	81.1 ± 0.4	93.0 ± 0.4	53.4 ± 1.4	**73.5 ± 0.5**	81.1 ± 0.8
Single-LLM implementation of multi-agent workflow						
AFlow (single-agent execution)	91.1 ± 1.6	78.8 ± 0.7	92.9 ± 0.1	53.8 ± 0.9	68.4 ± 0.1	81.1 ± 0.7
OneFlow (single-agent execution)	**92.1 ± 0.4**	**81.4 ± 0.6**	93.3 ± 0.1	54.1 ± 0.7	**73.5 ± 0.5**	81.7 ± 0.7

Table 2: Executor inference cost (USD) on public benchmarks with GPT-4o-mini. Values are mean ± std over three runs (lower is better). Best per column in **bold**; runner-up underlined.

	CODE		MATH		QA	
Method	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
Manual baselines						
CoT SC (5-shot)	$0.103 ± 0.000	$0.177 ± 0.000	$1.265 ± 0.001	$1.561 ± 0.009	$1.201 ± 0.001	$0.613 ± 0.001
MultiPersona	$0.099 ± 0.000	$0.226 ± 0.001	$0.429 ± 0.005	**$0.330 ± 0.021**	$0.415 ± 0.000	$0.301 ± 0.000
Automatically designed multi-agent frameworks						
AFlow	$0.198 ± 0.003	$0.393 ± 0.002	$1.134 ± 0.001	$2.343 ± 0.036	$1.438 ± 0.000	$0.771 ± 0.001
OneFlow	$0.026 ± 0.000	$0.070 ± 0.005	$0.623 ± 0.001	$0.819 ± 0.007	**$0.278 ± 0.000**	$0.322 ± 0.000
Single-LLM implementation of multi-agent workflow						
AFlow (single-agent execution)	$0.198 ± 0.004	$0.283 ± 0.001	$0.697 ± 0.001	$2.039 ± 0.028	$0.530 ± 0.001	$0.345 ± 0.001
OneFlow (single-agent execution)	**$0.020 ± 0.000**	**$0.063 ± 0.004**	**$0.387 ± 0.000**	$0.677 ± 0.002	**$0.278 ± 0.000**	**$0.284 ± 0.001**

one-time optimization cost and should be amortized over the intended deployment volume. To ensure statistical reliability, we conduct three independent trials and report mean values with standard deviations.

Baseline Methods. We compare against four categories of approaches: (1) *Manual baselines*: direct input-output (IO) prompting, chain-of-thought (CoT) (Wei et al., 2022), self-consistency, and MultiPersona (Wang et al., 2024); (2) *Automated multi-agent frameworks*: AFlow (Zhang et al., 2025e) and OneFlow; (3) *Heterogeneous multi-agent systems*: AFlow-optimized workflows using GPT-4o-mini and Claude-3.5-Haiku as heterogeneous executors; (4) *Single-LLM implementations*: to validate single-agent simulation (Section 3.2), we execute AFlow and OneFlow workflows using a single LLM agent.

Model Configuration. We use GPT-4o-mini as the primary executor with temperature 0. Both AFlow and OneFlow use Claude-4.0-Sonnet as the workflow designer (and critic for OneFlow) with 20 optimization rounds. For robustness, we additionally evaluate with Claude-3.5-Haiku and Qwen-3-8B (see Appendix). Heterogeneous workflow construction details are in Appendix A.9.

KV Cache Cost Measurement. OneFlow's KV-cache benefit is structural: collapsing a homogeneous workflow into one multi-turn conversation makes consecutive turns share a prefix, which is reused by the automatic prefix caching in modern LLM serving stacks. We do not implement custom cache management; both closed and open-weight runs go through the same request-level interface. For closed-weight executors (GPT-4o-mini, Claude-3.5-Haiku), the API does not expose prefill behavior, so we report ideal KV-cache costs estimated from the final conversation state under each provider's official tokenization. For Qwen-3-8B, we serve the model with vLLM and directly mea-

sure latency and throughput, complementing the token-count estimates.

4.2 Experimental Results and Analysis

4.2.1 Performance on public benchmarks

Single-agent execution is competitive with multi-agent performance. Table 1 summarizes results with GPT-4o-mini (pass@1 for code; F1 for QA; solve rate for math). Automatically designed workflows and their single-agent executions outperform manual baselines. Notably, executing AFlow- and OneFlow-designed workflows with *a single agent* often matches or slightly exceeds their multi-agent counterparts, consistent with the hypothesis in Section 3.1: in homogeneous settings, a single model can simulate agent roles via multi-turn conversations.

4.2.2 Cost on public benchmarks

Cost savings come from both workflow search and execution. Table 2 separates two sources of efficiency. First, for a fixed searched workflow, single-agent execution reduces cost through KV-cache reuse while maintaining comparable performance, as seen by comparing AFlow with AFlow (single-agent execution) and OneFlow with OneFlow (single-agent execution). Second, OneFlow is already cheaper than AFlow under the same multi-agent execution setting, reflecting the effect of cost-aware workflow search. Thus, the largest savings of OneFlow (single-agent execution) should be interpreted as the combined effect of a streamlined workflow discovered by OneFlow and single-agent execution with KV-cache reuse.

4.2.3 Pilot study on heterogeneous multi-agent workflows

A limited heterogeneous pilot shows mixed trade-offs. We include one automatically designed heterogeneous workflow with GPT-4o-mini and Claude-3.5-Haiku as executors (see Appendix A.1). Results are mixed: heterogeneity helps on some tasks, but does not consistently dominate homogeneous alternatives and can increase executor cost. In particular, the heterogeneous workflow attains the highest scores on HotpotQA (75.1) and MATH (55.7) in Table 3, while underperforming on HumanEval and offering no consistent gain elsewhere. We therefore treat this pilot as exploratory rather than evidence for a general ordering between homogeneous and heterogeneous workflows. In homogeneous settings, single-agent execution remains a strong, cost-efficient baseline, while systematic heterogeneous workflow design is left for future work.

4.2.4 Results on TravelPlanner and Qwen-3-8B

TravelPlanner (real-world planning and tool use). On TravelPlanner (Xie et al., 2024), single-agent execution of both AFlow and OneFlow matches the task success rate of their multi-agent counterparts while reducing executor cost. Figure 3 plots the cost–success trade-off: OneFlow workflows are visibly more compact than AFlow's, and OneFlow (single-agent execution) lies on the Pareto frontier. The benefits observed on academic benchmarks also carry over to planning-style tasks with longer trajectories and tool calls.

Open-weight executor (Qwen-3-8B with vLLM). The GPT-4o-mini cost numbers are token-count estimates, since the API hides prefill behavior. To obtain a directly measured latency signal under the same prefix-caching mechanism, we additionally run all methods on Qwen-3-8B served by vLLM (Table 4, Appendix A.2). On HumanEval, OneFlow reaches $87.0\%/87.4\%$ pass@1 under multi-/single-agent execution, comparable to AFlow's 86.8% under multi-agent execution; AFlow's single-agent variant scores 90.5%, about three points higher. Latency is more decisive: OneFlow runs at 4.3–4.8 s per sample, while AFlow takes $\sim$54 s in both execution modes, roughly an order-of-magnitude speedup from the streamlined workflow and shared prefix caching. This is consistent with the cost gap estimated for GPT-4o-mini.

5 Related Work

Multi-Agent Workflows and Task Decomposition. Multi-agent systems organize specialized LLM agents through sequential, iterative, or deliberative interactions to decompose complex tasks (Li et al., 2023; Zhu et al., 2025; Yao et al., 2022; Du et al., 2023). Single-LLM multi-persona methods such as Solo Performance Prompting/-MultiPersona (Wang et al., 2024) use one model to simulate multiple roles for task solving; our focus differs in that we study when an existing homogeneous workflow can be executed equivalently by one LLM instance, and how this affects workflow-level cost. Closest to our setting, Li (2026) studies when single-agent systems with skills can replace multi-agent systems and when they fail. Heterogeneous systems leverage model diversity (Ye et al.,

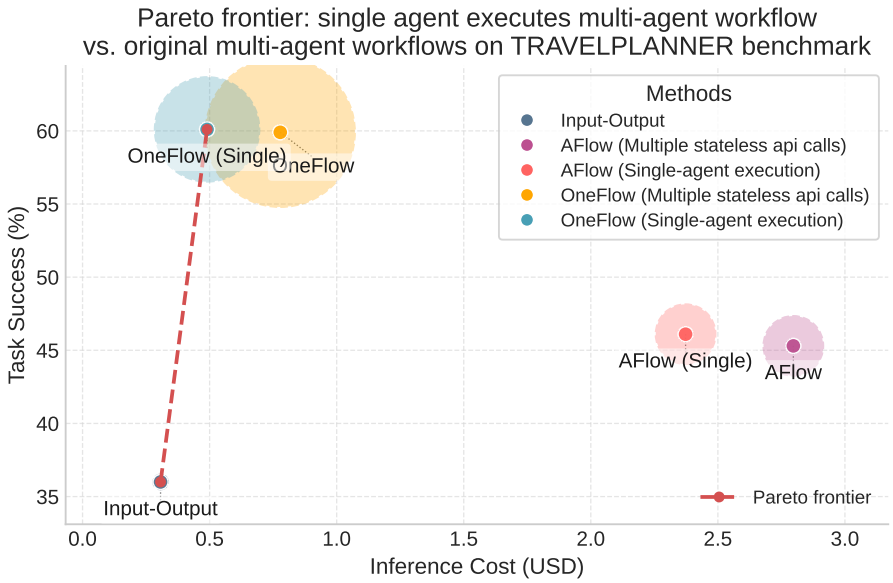

Figure 3: Pareto frontier on TravelPlanner (Xie et al., 2024): single-agent execution vs. multi-agent execution of homogeneous workflows. All workflows are searched by Claude-4.0-Sonnet and executed by GPT-4o-mini.

2025a; Zhang et al., 2025a; Wang et al., 2025b; Jiang et al., 2023; Chen et al., 2023), but typically emphasize ensembling or discussion rather than end-to-end execution with explicit control flow.

Automatic Design of Multi-Agent Workflows. A complementary line reduces human effort by automatically optimizing prompts (Khattab et al., 2024; Yuksekgonul et al., 2025), agent topology (Zhuge et al., 2024; Liu et al., 2024; Zhang et al., 2025c,d), or full workflows via search, fine-tuning, or meta-agents (Hu et al., 2025; Zhang et al., 2025e,b; Shang et al., 2025; Ye et al., 2025b; Gao et al., 2025; Wang et al., 2025c; Nie et al., 2025; Yue et al., 2025). Most assume homogeneous executors. Most closely, AFlow (Zhang et al., 2025e) performs MCTS-based workflow search and serves as our main automatic-workflow baseline; OneFlow adapts this framework with a cost-aware search objective and a single-agent execution mode. Our work complements this line by showing that a strong single-LLM simulator provides a competitive, cost-efficient baseline for many automatically designed homogeneous workflows.

KV Cache and Single-LLM Execution. In homogeneous workflows, a single-LLM simulator reuses KV cache across agent turns (Xiao et al., 2024), while heterogeneous workflows cannot share caches across different models. Recent work explores latent-space agent collaboration (Zou et al., 2025; Zheng et al., 2025; Fu et al., 2026) and structured caches for long or graph contexts (Wang et al., 2025a). Serving-level methods such as KVCOMM (Ye et al., 2026) share KV caches across contexts to reduce multi-agent prefill redundancy, which is complementary to our workflow-level approach and may narrow the cost gap when such infrastructure is available.

6 Conclusion

We investigate whether homogeneous multi-agent workflows can be effectively simulated by a single LLM agent. Under a scoped formulation with serialized execution and shared visible history, our analysis and experiments across seven benchmarks show that a single agent, through multi-turn conversations with KV cache reuse, can achieve comparable performance to multi-agent execution while substantially reducing inference cost. Building on this insight, we propose OneFlow, an automatic workflow design algorithm that optimizes for cost-efficiency, producing streamlined workflows compatible with single-agent execution. While single-LLM simulation cannot capture true model heterogeneity, our limited heterogeneous pilot shows mixed cost-performance trade-offs. Looking forward, we see training single agents for end-to-end workflow execution and developing principled methods for heterogeneous model composition as two complementary directions for advancing LLM multi-agent research.

Limitations

Our findings are scoped to homogeneous workflows under serialized execution with shared visible history. They do not claim equivalence for settings with private per-agent memory, asynchronous communication, parallel writes, or strategic and competitive interactions where controlled information asymmetry is central. Proposition 1 also relies on deterministic tools, history-visible routing, and aligned decoding randomness; relaxing these assumptions may introduce divergence between single-agent simulation and multi-agent execution. Empirically, our heterogeneous study is a pilot with automatically generated workflows and limited model combinations. These limitations delimit the claim but do not change the practical takeaway: in the targeted homogeneous setting, single-agent execution is a strong and cost-efficient baseline.

Ethical Considerations

This work evaluates LLM workflow designs on public benchmarks and a domain-specific shopping benchmark, without collecting new human-subject data. The main ethical risks are downstream misuse of more cost-efficient agentic systems, benchmark overfitting in automatic workflow search, and inaccurate or misleading outputs from LLM agents when deployed in real settings. We report the scope of our claims, avoid presenting the single-agent simulator as a replacement for heterogeneous or privacy-sensitive coordination settings, and recommend that practical deployments include task-appropriate validation, logging, and human oversight.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. 2023. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. *arXiv preprint arXiv:2309.13007*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. 2026. Cache-to-cache: Direct semantic communication between large language models. In *The Fourteenth International Conference on Learning Representations*.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. 2025. Flowreasoner: Reinforcing query-level meta-agents. *Preprint*, arXiv:2504.15257.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *12th International Conference on Learning Representations, ICLR 2024*.

Shengran Hu, Cong Lu, and Jeff Clune. 2025. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *Preprint*, arXiv:2306.02561.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, and 1 others. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

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 *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Xiaoxiao Li. 2026. When single-agent with skills replace multi-agent systems and when they fail. *arXiv preprint arXiv:2601.04748*.

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

Fan Nie, Lan Feng, Haotian Ye, Weixin Liang, Pan Lu, Huaxiu Yao, Alexandre Alahi, and James Zou. 2025. Weak-for-strong: Training weak meta-agent to harness strong executors. *arXiv preprint arXiv:2504.04785*.

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

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. 2025. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations*.

Haoyu Wang, Peihao Wang, Mufei Li, Shikun Liu, Siqi Miao, Zhangyang Wang, and Pan Li. 2025a. Graph-kv: Breaking sequence via injecting structural biases into large language models. *arXiv preprint arXiv:2506.07334*.

Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. 2025b. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*.

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*.

Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. 2025c. Scoreflow: Mastering llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2024. 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)*, pages 257–279, Mexico City, Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and 1 others. 2026. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:17882–17928.

Rui Ye, Xiangrui Liu, Qimin Wu, Xianghe Pang, Zhenfei Yin, Lei Bai, and Siheng Chen. 2025a. X-mas: Towards building multi-agent systems with heterogeneous llms. *Preprint*, arXiv:2505.16997.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. 2025b. MAS-GPT: Training LLMs to build LLM-based multi-agent systems. In *Forty-second International Conference on Machine Learning*.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. Masrouter: Learning to route llms for multi-agent systems. *arXiv preprint arXiv:2502.11133*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616.

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

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. 2025b. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025c. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2025d. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Forty-second International Conference on Machine Learning*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2025e. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*.

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. 2025. Thought communication in multiagent collaboration. *arXiv preprint arXiv:2510.20733*.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Yi Yao, Hanhao Li, Ningning Wang, Pai Liu, Tianhao Peng, Xin Gui, Xiaowan Li, Yuhui Liu, Yuchen Eleanor Jiang, Jun Wang, Changwang Zhang, Xiangru Tang, Ge Zhang, and 5 others. 2025. Oagents: An empirical study of building effective agents. *Preprint*, arXiv:2506.15741.

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, Louis Kirsch, Bing Li, Guohao Li, Shuming Liu, Jinjie Mai, Piotr Piękos, Aditya A. Ramesh, Imanol Schlag, Weimin Shi, and 7 others. 2025. Mindstorms in natural language-based societies of mind. *Computational Visual 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*.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, and 1 others. 2025. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*.

A Appendix

A.1 Executor-specific Results and Heterogeneous Baseline

A.2 KV Cache Reuse with Open-Weight LLMs

To validate findings beyond proprietary models, we run Qwen-3-8B with vLLM at 16k context. Single-agent execution of both AFlow and OneFlow maintains or improves pass@1 on HumanEval relative to multi-agent execution; although history accumulation increases input tokens (+967 for AFlow, +368 for OneFlow), latency/throughput remains stable due to KV-cache reuse.

A.3 The Use of Large Language Models

Large language models were used for sentence-level proofreading and LaTeX template adaptation. The research ideation, method design, experiments, and scientific claims were conducted by the authors.

A.4 Monte Carlo Tree Search for Workflow Optimization

As mentioned above and Figure 2, we employ MCTS to systematically explore the space of possible workflows, treating each workflow configuration as a node in the search tree. Specifically:

1. Selection: At each iteration, we form a candidate set consisting of the initial workflow $W_{\text{Input-Output}}$ together with the top 3 best-performing workflows from the Monte Carlo Tree. Retaining the initial workflow $W_{\text{Input-Output}}$ in the candidate set helps the framework maintain exploration of the workflow space. We then select one Workflow W from the candidate set for expansion. Following AFlow (Zhang et al., 2025e), we employ a mixed-probability selection strategy that combines uniform distribution (for exploration) and score-based weighting (for exploitation):

$$p_i = \lambda \cdot \frac{1}{n} + (1 - \lambda) \cdot \frac{\exp(\alpha \cdot s_i)}{\sum_j \exp(\alpha \cdot s_j)} \quad (2)$$

where s_i is the score of workflow i, α controls the sharpness of the distribution, λ balances exploration and exploitation, and $n \leq 4$ represents the number of candidates. This formula ensures that better-performing workflows have a higher probability of being selected as Workflow W.

2. Expansion: This phase leverages the dual meta-LLM architecture (detailed in Appendix A.5).

Table 3: Executor-specific results and heterogeneous baseline on public benchmarks. Values are mean ± std over three runs, in percentage (0–100). Best per column in **bold**; best per base model type per column underscored.

	CODE		MATH		QA	
Method (Executor)	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
GPT-4o mini-based						
AFlow (GPT-4o mini)	90.1 ± 0.0	78.8 ± 0.7	**93.6 ± 0.5**	55.6 ± 0.3	72.1 ± 0.2	83.1 ± 0.3
OneFlow (GPT-4o mini)	91.6 ± 0.8	81.1 ± 0.4	93.0 ± 0.4	53.4 ± 1.4	73.5 ± 0.5	81.1 ± 0.8
AFlow (GPT-4o mini, single-agent)	91.1 ± 1.6	78.8 ± 0.7	92.9 ± 0.1	53.8 ± 0.9	68.4 ± 0.1	81.1 ± 0.7
OneFlow (GPT-4o mini, single-agent)	**92.1 ± 0.4**	81.4 ± 0.6	93.3 ± 0.1	54.1 ± 0.7	73.5 ± 0.5	81.7 ± 0.7
Claude 3.5 Haiku-based						
AFlow (Claude 3.5 Haiku)	90.8 ± 0.0	83.6 ± 0.0	91.2 ± 0.0	50.5 ± 0.0	74.6 ± 0.0	86.8 ± 0.0
OneFlow (Claude 3.5 Haiku)	91.6 ± 0.0	**84.4 ± 0.0**	93.0 ± 0.0	51.3 ± 0.0	74.7 ± 0.0	**87.5 ± 0.0**
Heterogeneous baseline						
AFlow (Heterogeneous: GPT-4o mini + Claude 3.5 Haiku)	87.0 ± 0.8	80.0 ± 0.3	**93.6 ± 0.3**	**55.7 ± 0.6**	**75.1 ± 0.5**	85.5 ± 0.5

Table 4: Results on HumanEval with Qwen-3-8B. Single-agent execution maintains performance and efficiency thanks to KV cache reuse. Scores are averaged over three independent runs.

Method	pass@1	Avg Latency (s)	Throughput (samples/s)	Avg Input Tokens	Avg Output Tokens
CoT	83.5%	2.60	7.66	213	74
CoT SC × 5	83.7%	17.44	1.32	1748	502
MultiPersona	84.7%	32.38	0.75	1722	846
AFlow (multi-agent execution)	86.8%	54.98	0.17	2302	1753
AFlow (single-agent execution)	90.5%	53.53	0.18	3269	1739
OneFlow (multi-agent execution)	87.0%	4.31	2.47	920	148
OneFlow (single-agent execution)	87.4%	4.83	1.70	1288	159

At each expansion, the designer meta-LLM proposes a new workflow design W_{new1} (represented in Python code with detailed comments) based on the selected workflow W from the previous selection step and the failed cases in the validation dataset D_v. Next, the critic meta-LLM reviews the proposed workflow design W_{new1} to assess its validity and efficiency by examining the code comments and Python implementation. The critic also compares the proposal W_{new1} with candidate workflows from the selection stage. Finally, the critic meta-LLM proposes an improved workflow W_{new2} based on the designer's proposal W_{new1}, writing detailed comments in the workflow code to record changes and their rationale. The output of this stage is a runnable workflow.py file that implements W_{new2}.

3. Evaluation. The proposed workflow W_{new2} is then evaluated on the validation dataset D_v to obtain performance metrics (e.g., accuracy), cost metrics (e.g., token consumption), and failed samples (including the specific reasoning processes of the failures).

4. Backpropagation: The performance and cost, and failure cases are stored in the current workflow node W_{new2}. The relative success or failure compared to the parent workflow node W is stored in the parent node W (we call this backpropagation) to avoid proposing similar workflows. The optimization process continues until a maximum number of iterations is reached (20).

A.5 Dual Meta-LLMs for Balanced Performance and Cost Optimization

We design two specialized meta-LLMs that collaboratively design new workflows (the right panel of Figure 2 shows how this dual meta-LLM framework works):

Meta-LLM 1: Creative Designer. This meta-LLM's goal is to creatively, even disruptively, improve a workflow. It receives the code and prompt of the current agentic workflow W_{current}, the workflow's performance score (e.g., accuracy), any previous modifications based on W_{current}, the corresponding performance score, and a sample of incorrectly answered questions, D_{error}, including the questions, the reasoning process when the workflow failed to answer the question, and the ground truth answer. It also receives instructions on how to write a runnable `workflow.py` file, how to define

system prompts for LLM agents, and how to utilize existing operators (e.g., chain-of-thought reasoning, ensembling, executing Python code, etc.). With this information, the Creative Designer is prompted to make creative modifications to improve workflow performance and add comments to the workflow code explaining the rationale for the changes.

$$W_{\text{creative designer}} = \text{Designer}(W_{\text{current}}, D_{\text{error}}, \text{instructions}) \quad (3)$$

The creative designer's setting is similar to the AFlow (Zhang et al., 2025e) setting, with two modifications: 1) the creative designer is prompted to write code comments explaining the rationale for modifications, which can inform the other meta-LLM (the critical reviewer), and 2) the creative designer is given more detailed error logs, which include the specific reasoning process during the entire workflow's execution. This helps the creative designer better understand the workflow's failure cases. Empirically, we found that the creative designer alone can propose workflows with excellent performance, but the inference cost of the workflow is high.

Meta-LLM 2: Critical Reviewer. The Critical Reviewer's goal is to review and critique the creative designer's proposed workflow and modify it to avoid mistakes and improve cost-efficiency. To accomplish this, we provide the critical reviewer with the same information as the creative designer. Additionally, we give the critical reviewer the creative designer's proposed workflow (a runnable `workflow.py` file, including code and comments) and the cost and performance metrics of existing workflow candidates (the top 3 best-performing workflows from the Monte Carlo tree, plus the initial workflow $W_{\text{Input-Output}}$). We then prompt the critical reviewer to carefully examine the creative designer's proposed workflow, using the existing workflow candidates' cost and performance as reference points. The critical reviewer proposes a new workflow design based on the designer's proposal, writing detailed comments in the workflow code to document changes and their rationale. The goal is to avoid mistakes and improve cost-efficiency. The output of this stage is a runnable `workflow.py` file. The critical reviewer also writes reflections on the improved workflow. Since the critical reviewer is given more global information and cost data, it can provide a broader view of the workflow's performance and cost trade-offs to guide future rounds of workflow design.

$$W_{\text{critical reviewer}} = \text{Reviewer}(W_{\text{creative designer}}, W_{\text{candidates}}, \text{cost_metrics}) \quad (4)$$

We use the critical reviewer's output workflow as the final workflow of this round.

A.6 Results on Claude 3.5 Haiku

To further evaluate the effectiveness of our proposed OneFlow framework, we conduct experiments on Claude 3.5 Haiku. The results are shown in Table 5 and Table 6.

Performance on Shopping-Specific Tasks. Table 7 shows that OneFlow obtains the best result on 9 out of 10 shopping-specific tasks against both single-LLM baselines and existing multi-agent frameworks. We observe improvements over the strongest baseline (AFlow) on Product Selection (+4.0%, 0.678 vs. 0.638), Sentiment Analysis (+2.5%, 0.777 vs. 0.752), and Multilingual Query Understanding (+4.8%, 0.569 vs. 0.521).

A.7 More results on the cost analysis

Per-executor inference costs for GPT-4o-mini are reported in Table 2 (main text), and for Claude-3.5-Haiku in Table 6. The AFlow heterogeneous baseline (GPT-4o-mini + Claude-3.5-Haiku) incurs executor costs of \$0.278, \$0.343, \$1.469, \$1.334, \$2.153, and \$0.822 on HumanEval, MBPP, GSM8K, MATH, HotpotQA, and DROP respectively, generally between the homogeneous GPT-4o-mini and Claude-3.5-Haiku settings.

Table 9 reports the underlying input and output token counts for GPT-4o-mini, broken out by execution mode. Single-agent execution reduces input tokens for the AFlow workflow (e.g., $2863 \rightarrow 2597$ on HumanEval, $8501 \rightarrow 5572$ on HotpotQA), reflecting prefix sharing across agent turns. OneFlow's input tokens are already much lower than AFlow's under multi-agent execution, and decrease further under single-agent execution.

A.8 System prompt for OneFlow

A.8.1 System prompt for the creative designer Meta-LLM

The following designer prompts are adapted from AFlow (Zhang et al., 2025e).

Table 5: Results on Claude 3.5 Haiku. Performance comparison on public benchmarks. We report pass@1 accuracy for code generation tasks (HumanEval, MBPP) and F1 scores for question-answering tasks (HotpotQA, DROP), solve rate (%) for GSM8K, MATH. OneFlow achieves competitive performance across most benchmarks. Results are averaged over three independent runs.

Method	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
IO	0.870	0.745	0.875	0.329	0.635	0.640
AFlow (Zhang et al., 2025e)	0.908	0.836	0.912	0.505	0.746	0.868
OneFlow (Ours)	**0.916**	**0.844**	**0.930**	**0.513**	**0.747**	**0.875**

Table 6: Results on Claude 3.5 Haiku. Executor inference cost comparison on public benchmarks. Values are in USD (lower is better). OneFlow achieves lower executor costs on most benchmarks while maintaining competitive performance. Results are averaged over three independent runs.

Method	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
AFlow (Zhang et al., 2025e)	**0.20**	2.39	6.68	**3.55**	6.55	3.46
OneFlow (Ours)	0.25	**0.58**	**5.83**	4.33	**1.47**	**3.14**

Table 7: Performance comparison on Shopping-MMLU tasks using accuracy as the evaluation metric. OneFlow achieves the best result among evaluated methods on 9 out of 10 shopping-specific tasks. Results are averaged over three independent runs. Task abbreviations: Product Sel. (applicable product selection), Sentiment (aspect-based sentiment classification), Attribute (implicit attribute selection), Multi-lang (multilingual query product semantic understanding), Prod. Comp. (product complements), Query (query product semantic classification), Brand (related brands selection), Keyword (related keyword intent selection), Rev. Help (review helpfulness selection), Rev. Sent. (reviews overall sentiment selection). Best results are shown in **bold**.

Method	Product Sel.	Sentiment	Attribute	Multi-lang	Prod. Comp.	Query	Brand	Keyword	Rev. Help	Rev. Sent.
IO	0.638	0.668	0.650	0.465	0.713	0.464	0.681	0.728	0.586	0.578
CoT (Wei et al., 2022)	0.647	0.712	0.700	0.441	0.700	0.455	**0.761**	0.713	0.586	0.566
SC(CoT X 5) (Wang et al., 2023)	0.656	0.731	0.713	0.473	0.703	0.491	0.761	0.725	0.598	0.566
AFlow (Zhang et al., 2025e)	0.638	0.752	0.769	0.521	0.736	0.530	**0.761**	0.713	0.550	0.698
OneFlow (Ours)	**0.678**	**0.777**	**0.775**	**0.569**	**0.756**	**0.567**	0.756	**0.783**	**0.609**	**0.718**

Table 8: Cost efficiency comparison on Shopping-MMLU tasks measured in USD executor inference cost (lower is better). Results are averaged over three independent runs. OneFlow demonstrates superior cost efficiency across all shopping-specific tasks while maintaining competitive performance. Best results are shown in **bold**.

Method	Product Sel.	Sentiment	Attribute	Multi-lang	Prod. Comp.	Query	Brand	Keyword	Rev. Help	Rev. Sent.
SC(CoT X 5) (Wang et al., 2023)	1.87	1.78	1.62	1.84	1.89	1.55	1.34	2.00	1.53	2.78
AFlow (Zhang et al., 2025e)	1.47	1.83	2.20	1.32	1.39	0.63	**0.23**	**0.39**	0.86	2.75
OneFlow (Ours)	**0.26**	**0.48**	**0.63**	**0.46**	**0.42**	**0.47**	0.32	0.60	**0.31**	**0.74**

Table 9: Executor-specific token usage on public benchmarks (GPT-4o-mini). Left: input tokens; right: output tokens. Values are averaged over three runs.

	Input tokens						Output tokens					
Method	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP	HumanEval	MBPP	GSM8K	MATH	HotpotQA	DROP
AFlow (multi-agent)	2863	2049	2738	10793	8501	4003	1880	1409	1107	5361	871	605
OneFlow (multi-agent)	488	586	1062	2096	1559	1191	205	196	719	2286	190	374
AFlow (single-agent)	2597	414	1044	9382	5572	1084	1875	1279	840	4979	655	447
OneFlow (single-agent)	313	443	215	2164	1559	1134	174	196	557	1782	190	307

```
WORKFLOW_OPTIMIZE_PROMPT_DESIGNER = """You
    are building a workflow Graph (nodes are
     llm agents, edges are the flow of
    information) and corresponding Prompt to
     jointly solve {type} problems.
Referring to the given graph and prompt,
    which forms an example of a {type}
    solution approach, please reconstruct
    and optimize them.
You can add, modify, or delete nodes, edges,
     or prompts. Include your single
    modification in XML tags in your reply.
    Ensure they are complete and correct to
    avoid runtime failures.
When optimizing, you can incorporate
    critical thinking methods such as review
    , revise, ensemble (generating multiple
    answers through different/similar
    prompts, then voting/integrating/
    checking the majority to obtain a final
    answer), brainstorming, expert-based
    reasoning, to solve the problem
    efficiently and effectively.
Consider Python's loops (for, while, list
    comprehensions), conditional statements
    (if-elif-else, ternary operators), and
    other programming techniques to enhance
    the workflow. Use logical and control
    flow (IF-ELSE, loops) for a more
    enhanced graphical representation.
You can design a workflow that adapts its
    approach based on the complexity of each
     problem. For example, using Python if-
    else statements to choose different
    solution strategies for easy vs. hard
    problems.

PROMPT REQUIREMENTS:
- Only generate prompts used by Custom
    operators in your graph (accessed as
    prompt_custom.YOUR_PROMPT_NAME)
- Built-in operators already have their own
    prompts - don't generate prompts for
    them
- Remove any unused prompts from
    prompt_custom
- Generated prompts must be complete with no
     placeholders

Output the modified graph and all necessary
    prompt_custom prompts.
It's crucial to include necessary context
    during the process.
Be creative and try to push the boundaries
    of the existing graph.
Write concise and informative inline
    comments for the code you modified to
    explain the logic and the purpose, why
    the modification is creative and
    effective. With all the comments you
    write, you should leave your name:
    Designer (modifying the round X graph).
If you encounter code comments written by
    other designers or critics, keep those
    that are useful for future reference.
"""


WORKFLOW_INPUT_DESIGNER = """
Here is a workflow graph and the
    corresponding prompt (prompt only
    related to the custom method) that has
    been tested on the previous round and
    its score (maximum score is 1). You must
     make further optimizations and
    improvements based on this graph. The
    modified graph must differ from the
    provided example, and the specific
    differences should be noted within the <
    modification>xxx</modification> section
    .\n
<sample>
    <experience>{experience}</experience>
    <modification>(such as:add /delete /
    modify/ ...)</modification>
    <score>{score}</score>
    <graph>{graph}</graph>
    <prompt>{prompt}</prompt>(contains
    prompts used by Custom operators)
    <operator_description>{
    operator_description}</
    operator_description>
</sample>

You are encouraged to use the operators
    provided, not limited to the custom
    operator.


In the graph, you should use the list
    process to record the intermediate
    output of the workflow. And return the
    process as the one of the output of the
    workflow.
Below are the logs of some results with the
    aforementioned workflow Graph
    encountered errors, which can be used as
     references for optimization:
{log}

In your design, you cannot hardcode the
    answer (e.g., remember the answer of a
    specific problem), which can lead to
    overfitting. Instead, provide principles
```

```
     rather than specific answers.
First, provide optimization ideas. **Only
    one detail point can be modified**, and
    no more than 5 lines of code may be
    changed per modification; you can not
    totally change the existing graph,
    extensive modifications are strictly
    prohibited to maintain project focus!
When introducing new functionalities in the
    graph, please make sure to import the
    necessary libraries or modules yourself,
     except for operator, prompt_custom,
    create_llm_instance, which have already
    been automatically imported.
No need to import the operator,
    prompt_custom, create_llm_instance,
    which have already been automatically
    imported.

**Under no circumstances should Graph output
     None for any field.**
Use self.custom methods to restrict your
    output format, rather than using code (
    outside of the code, the system will
    extract answers based on certain rules
    and score them).
It is very important to format the Graph
    output answers, you can refer to the
    standard answer format in the log.
Be creative and try your best to propose new
     ideas.

"""
```

A.8.2 System prompt for the critical reviewer Meta-LLM

The following critic prompts are adapted from AFlow (Zhang et al., 2025e).

```
WORKFLOW_OPTIMIZE_PROMPT_CRITIC = """You are
     a great workflow critic and optimizer.
You are the analytical critic and practical
    optimizer. Focus on (1) the potential of
     the designer's idea to push the
    boundaries of the existing workflow
    solutions and (2) cost-effectiveness of
    the designer's proposed solution.
- The designer's idea has a great chance to
    help us push the boundaries of the
    existing graph and may find great
    solutions, so you should try to
    implement the designer's idea in a cost-
    efficient way, if necessary.
- Preserve the designer's innovative ideas
    while making them more cost-efficient,
    if necessary.
- You are encouraged to use the operators
    provided, not limited to the custom
    operator.
- Fully respect the designer's idea,
    especially when the designer proposed to
     use a predefined operators, you just
    use it and do not modify it. Some
    operators can be expensive, but if the
    designer proposed to use them, you
    should give them a try as it may have
    great potential of performance
    improvement.
- If a solution involves too many api calls
    (agent) e.g., > 5 api calls per problem
    and are not necessary, it can be very
    slow and costly, you can try to find a
    way to make it more cost-efficient.
    Overly expensive solutions are not
    allowed.
- You have access to the existing top
    performing workflow graphs and their
    performance and cost metrics. Consider
    the performance and cost of these
    existing graphs (they are the existing
    solutions for the same problem) when
    optimizing the designer's graph. You
    should not blindly reuse the existing
    top performing graphs, but use them as
    references to improve the cost-
    efficiency of the designer's idea, which
     may have great potential.

## Code Commentary Requirements:
- Add concise inline comments explaining
    your critical analysis and optimization
    rationale
- Include practical critiques at the code's
    end, drawing from top-performing graphs
    and best practices
- Preserve useful existing comments from
    other designers/critics, remove
    redundant ones
- Sign your comments as: "Critic (modifying
    round X workflow proposed by the
    designer)"

## Context and References:
When optimizing, refer to the designer's
    comments to understand the logic,
    purpose, and creative value of each
    modification. Consider how your
    optimization maintains the innovation
    while improving practical efficiency.

Referring to the designer's given graph and
    prompt for {type} problems, optimize
    them following the framework above."""




# Critic input prompts
WORKFLOW_INPUT_CRITIC = """
Here is a workflow graph and the
    corresponding prompt (prompt only
    related to the custom method) that has
    been proposed by a creative designer LLM
     (based on one existing workflow), which
     can be a great idea. You must respect
    and implement the designer's ideas. If
    the designer LLM's proposed workflow is
    already cost-effective, you just use it
    and do not change it. When the designer'
    s idea is costly, you need to make
    further optimizations based on this
    graph to make it more cost-efficient.
    Try to preserve the original ideas as
    much as possible, but implement them in
    a great cost-efficient way if necessary.
     You are also provided with the existing
     top performing workflow graphs, their
```

```
    prompts, their performance metrics, most
     importantly, their cost metrics, on the
     same problems. Critically learn from
    their cost and strategy, and focus on
    improving the cost-efficiency of the
    designer's idea. Your improved workflow,
     by nature, is a great and cost-
    efficient implementation of the designer
    's idea.

{experience}

Only when you are very sure that the
    designer's idea is not good based on
    strong evidence, you can propose a new
    idea or fundamental changes. But in most
     cases, you should try to improve the
    designer's idea's cost-efficiency while
    preserving the original ideas, which can
     help us utilize the designer's
    innovative idea to push the boundaries
    of the existing workflow graph and find
    great solutions. The modified graph must
     differ from the provided graph (only
    exception is that, If the designer LLM's
     proposed workflow is already cost-
    effective, you just use it and do not
    change it.), and the specific
    differences should be noted within the <
    modification>xxx</modification> section,
     here is the workflow graph proposed by
    the designer LLM.
<sample>
    <modification>(such as:add /delete /
    modify/ ...)</modification>
    <graph>{graph}</graph>
    <prompt>{prompt}</prompt>(contains
    prompts used by Custom operators)
    <operator_description>{
    operator_description}</
    operator_description>
</sample>

In the graph, you should use the list
    process to record the intermediate
    output of the workflow. And return the
    process as the one of the output of the
    workflow.

You are encouraged to use the operators
    provided, not limited to the custom
    operator.

**ERROR LOGS FROM THE ORIGINAL WORKFLOW:**
The designer proposes an improved workflow (
    the graph above) based on an existing
    graph. The error logs of the old
    workflow that the designer is improving
    are:
{log}

**IMPORTANT:** Pay close attention to the
    error logs above. These logs show
    specific issues, failures, and problems
    encountered by the old workflow. Use
    these error logs to evaluate whether the
     designer's proposed improvements have
    potential to address these specific
    issues. In your design, you cannot
    hardcode the answer (e.g., remember the
    answer of a specific problem), which can
     lead to overfitting. Instead, provide
    principles rather than specific answers.

**CRITICAL FORMATTING RULE - READ THIS FIRST
    :**
NEVER combine formatting instructions with
    reasoning instructions in the same
    prompt. If you need both reasoning AND
    specific output formatting, you MUST use
     separate agents or code-based
    formatting.

**WHY:** LLMs cannot simultaneously reason
    deeply AND maintain strict output
    formats. When you ask an LLM to "think
    about this problem AND format your
    answer as X", it will prioritize
    thinking over formatting, causing format
     failures.

**SOLUTION:** Use one agent for reasoning,
    then use code (regex) or a separate
    formatting agent to ensure correct
    output format. This is non-negotiable
    for reliable results.


When introducing new functionalities in the
    graph, please make sure to import the
    necessary libraries or modules yourself,
     except for operator, prompt_custom,
    create_llm_instance, which have already
    been automatically imported.
No need to import the operator,
    prompt_custom, create_llm_instance,
    which have already been automatically
    imported.

**Under no circumstances should Graph output
     None for any field.**
Use self.custom methods to restrict your
    output format, rather than using code (
    outside of the code, the system will
    extract answers based on certain rules
    and score them).


Here are the existing top performing
    workflow graphs (the round 1 is not the
    top solution, it is a baseline solution)
     for the same problem and their
    performance and cost metrics. Do not
    blindly reuse the existing top
    performing graphs, but pay attention to
    their cost and performance, and use them
     as references to improve the cost-
    efficiency of the designer's idea if
    needed:
{top_solutions_context}
"""
```

A.9 System prompts for AFlow-optimized heterogeneous multi-agent systems

The following guidance is adapted from AFlow (Zhang et al., 2025e).

```
# Optional guidance snippet appended when
    running in heterogeneous executor mode
HETERO_EXEC_GUIDANCE = """
\n[HETEROGENEOUS EXECUTOR MODE]
You can use two different executor LLMs
    within a single workflow: `claude-3-5-
    haiku-20241022` and `gpt-4o-mini`.
- Define your Workflow __init__ to accept
    two configs (e.g., llm_config_haiku35
    and llm_config_4omini), create two
    Custom operators (self.custom_haiku35,
    self.custom_4omini), and choose between
    them per step based on problem
    characteristics.
- IMPORTANT: Do not rely on a single
    executor. Prefer designs that leverage
    BOTH executors creatively within the
    same workflow (e.g., draft with 4o-mini
    -> verify/refine with Haiku; brainstorm
    with Haiku -> select/format with 4o-mini
    ; parallel generate-and-vote using both,
     etc.).
- If you decide to use only one executor for
     a step, briefly justify why the other
    is not helpful for that specific step (
    cost/perf tradeoff). Over the whole
    workflow, ensure both executors have
    meaningful roles.
- Keep each round's modification minimal.
    Maintain imports to allowed modules and
    record intermediate steps in `process`.
- Prompts you generate still belong to the `
    prompt_custom` module and are used by
    Custom operators regardless of which
    executor calls them.
"""
```

A.10 Details about the datasets

Dataset Selection and Evaluation Protocol. We use the same six datasets as AFlow (Zhang et al., 2025e): GSM8K, HumanEval, MBPP, HotpotQA, DROP, and MATH. The number of testing samples in each dataset are: HumanEval (131), MBPP (341), MATH (486), GSM8K (1055), HotpotQA (800), and DROP (800). We also evaluate on TravelPlanner (Xie et al., 2024), a benchmark for real-world planning and tool use with Language Agents. The number of testing samples in TravelPlanner are: TravelPlanner (180).

Shopping-MMLU: Domain-Specific Evaluation. To assess performance on domain-specific reasoning tasks, we introduce evaluation on Shopping-MMLU, a specialized benchmark for e-commerce reasoning capabilities. Our evaluation protocol for Shopping-MMLU follows a rigorous difficulty-based selection process. We initially evaluated all 33 shopping-related multiple-choice questions using Claude 3.5 Sonnet as a screening model. From these initial evaluations, we identified and selected the 10 most challenging tasks where Claude 3.5 Sonnet achieved less than 80% accuracy, ensuring our evaluation focuses on genuinely difficult reasoning scenarios that require sophisticated multi-agent collaboration.

The selected Shopping-MMLU tasks span e-commerce domains including product selection, sentiment analysis, attribute reasoning, multilingual query understanding, product complementarity, semantic classification, brand relationships, keyword intent analysis, review helpfulness assessment, and sentiment evaluation. Results are reported in Table 7.