AGENTASK: Multi-Agent Systems Need to Ask

Bohan Lin[⋆], **Kuo Yang**[⋆], **Yingchuan Lai**[♣], **Yudong Zhang**[⋆✉], **Chen Zhang**[⋆], **Guibin Zhang**[♠], **Xinlei Yu**[♠], **Miao Yu**[⋆], **Xu Wang**[⋆], and **Yang Wang**[⋆✉]
[⋆]University of Science and Technology of China
[♣]Xi'an Jiaotong University [♠]National University of Singapore
Contact: linbohan@stu.xjtu.edu.cn, zyd2020@mail.ustc.edu.cn
✉ Corresponding authors

Abstract

Multi-agent systems built on large language models (LLMs) promise enhanced problem-solving capabilities through collaborative division of labor. However, they frequently underperform single-agent baselines due to *edge-level* error cascades: minor inaccuracies at one message handoff propagate across the entire chain. We propose **AgentAsk**, a lightweight and plug-and-play clarification module that treats every inter-agent message as a potential failure point and inserts *minimally necessary* questions to arrest error propagation. AgentAsk follows a three-stage pipeline: (i) distilling edge-level judgments from curated failure traces into a compact policy, (ii) supervising the policy to determine *when/what/whom/how* to ask, and (iii) optimizing online with **E-GRPO**, a reinforcement learning objective that balances accuracy, latency, and cost. The module is architecture-agnostic and easy to integrate into existing orchestration. Across math, reasoning, and coding benchmarks, AgentAsk consistently improves accuracy and robustness over public multi-agent implementations while keeping overhead minimal, with latency and extra cost all less than 5%, approaching the performance of a strong evaluator. Beyond empirical improvements, we contribute a principled taxonomy of edge-level errors and a practical recipe for link-local intervention, offering a scalable pathway toward more reliable LLM-based multi-agent systems. The code is available at https://github.com/BoHan-LIN04/AgentAsk.

1 Introduction

Large language model (LLM)–based agent systems have attracted increasing attention for their ability to combine multi-step reasoning with dynamic tool use (Wei et al., 2022a; Schick et al., 2023). By coordinating multiple LLM-driven agents, these systems aim to leverage collective intelligence to solve complex, real-world tasks that necessitate decomposition, external knowledge access, and iterative refinement(Park et al., 2023). This paradigm is referred to as a multi-agent system (MAS). Applications now span diverse domains such as software engineering (Hong et al., 2024; Qian et al., 2024) and AI for Science (Ghafarollahi and Buehler, 2025; M. Bran et al., 2024), and there is growing interest in developing general-purpose multi-agent frameworks for cross-domain problem solving (Wu et al., 2024a; Liu et al., 2024; Yu et al., 2025a). These advances underscore both the promise of multi-agent collaboration and the difficulty of making MAS reliable in practice.

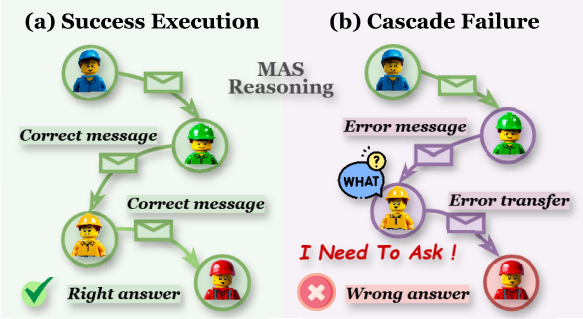


Figure 1: **Multi-Agent System Reasoning: Success vs. Cascade Failure.** The left shows the normal operation of MAS, while the right shows that the upstream agents transmit errors during the interaction that trigger a cascade effect and cause the entire system to fail.

Despite the promise of collaborative decomposition, multi-agent systems often fail to reliably surpass strong single-agent baselines across diverse tasks. Previous research and benchmarks report unstable gains and brittle behavior under realistic orchestration (Liu et al., 2024; Cemri et al., 2025). Execution logs reveal a recurring pattern in which small inconsistencies or missing details introduced at one step travel along the chain and accumulate into system-level failures. Building on this pattern, chainwise amplification is intensified by specification gaps and inter-agent misalignment, and in some settings, collaboration reduces accuracy (Zhang et al., 2025d; Wynn et al., 2025; Wang

et al., 2025). Prior studies quantify the effect, with success on difficult programming tasks falling to about 25% (Cemri et al., 2025). Similar vulnerabilities appear in single-agent traces, including misinterpretations, logical gaps, and limited reflection, which indicates that subtle errors arise early and persist through execution. This sets up the problem we study: limiting error growth at the handoff between agents so that small inconsistencies do not accumulate into system-level failures.

A growing body of recent research attempts to improve the reliability of MAS. One direction emphasizes structured roles and workflow governance, enforcing explicit hierarchies and protocols to reduce miscommunication (Hong et al., 2024; Qian et al., 2024; Chen et al., 2024b; Yu et al., 2025b). Another line refines orchestration and prompting strategies, leveraging prompt engineering and search to better allocate tasks (Zhuge et al., 2024; Zhang et al., 2025b,a; Yue et al., 2025). Concurrently, self-checking and feedback mechanisms encourage agents to reflect on or repair their own outputs (Liang et al., 2024; Shinn et al., 2023; Madaan et al., 2023). These approaches, spanning architecture, algorithm, and introspection, provide useful gains but remain limited. Many proposed solutions are domain-specific, costly to scale, or increase complexity without offering general principles. Moreover, existing taxonomies of failure are largely descriptive and rarely yield prescriptive mechanisms for prevention or correction. Consequently, current systems remain fragile when confronted with complex or open-ended tasks.

We argue that ensuring reliability necessitates a paradigm shift toward *edge-level* intervention: each inter-agent message should be treated as a potential failure point where minimal clarification can halt cascading errors, illustrated in Figure 1. To this end, we propose **AgentAsk**, a plug-and-play clarification module for LLM-based multi-agent systems. AgentAsk introduces a taxonomy of four core edge-level error types and implements a three-stage pipeline: (i) distilling knowledge from failure traces with a powerful evaluator, (ii) transferring this distilled capability to a lightweight clarifier trained to generate targeted questions, and (iii) refining the clarifier through reinforcement learning for adaptive, budget-aware behavior. Crucially, AgentAsk is designed to be lightweight, architecture-agnostic, and easily integrated into existing frameworks, enabling systems to intercept misunderstandings before they spread.

Our work makes the following contributions:

❶ ***Edge-level perspective and taxonomy:*** We formalize a four-type taxonomy of inter-agent errors and motivate clarification as a first-class mechanism for multi-agent collaboration.

❷ ***Practical clarification module:*** **AgentAsk** is a lightweight and plug-and-play module trained with **E-GRPO** reinforcement learning method, which agents can seamlessly integrate to identify and resolve interaction errors.

❸ ***Comprehensive empirical evaluation:*** Extensive experiments across diverse benchmarks demonstrate that AgentAsk consistently improves both accuracy and robustness over public multi-agent baselines with excessive latency and extra cost of only less than 5%.

2 Related Work

LLM-Based Multi-Agent Systems. MAS coordinates multiple role-specialized agents, often equipped with tools, to solve complex tasks. Early systems demonstrated that organizing multiple LLMs into role-specialized, conversational teams with tool use can outperform single models under controlled settings, through frameworks that scaffold roles, protocols, and conversation programming such as AutoGen (Wu et al., 2024a), AgentVerse (Chen et al., 2024b), CAMEL (Li et al., 2023), MetaGPT (Hong et al., 2024), and ChatDev (Qian et al., 2024). Beyond handcrafted blueprints, recent work treats orchestration as a learnable object: GPTSwarm models agents and communication links as an optimizable graph (Zhuge et al., 2024), AFlow searches over code-represented workflows via Monte Carlo tree search (Zhang et al., 2025b), MaAS samples query-dependent sub-architectures from an agentic supernet (Zhang et al., 2025a), and MasRouter learns a cascaded controller for collaboration mode, role allocation, and LLM selection (Yue et al., 2025). These directions build on and complement decision-time tool use, such as ReAct (Yao et al., 2023) and Toolformer (Schick et al., 2023). Our work is orthogonal to global orchestration search: we hold an existing pipeline fixed and inject a lightweight edge-level clarifier that adds minimal, well-placed questions at message handoffs to curb error propagation with low latency and cost.

Failure Analyses in MAS. Large-scale audits show that multi-agent gains over strong single-agent baselines are not guaranteed and that many

failures arise at message handoffs where missing details, ambiguous references, or distorted intermediate results are left unclarified (Cemri et al., 2025; Zhu et al., 2025; Zhang et al., 2025d). Methodologically related attempts to improve reliability include multi-agent debate and discussion (Chan et al., 2024; Liang et al., 2024), self-feedback loops such as Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023), and verification or self-correction procedures (Wu et al., 2024b; Lee et al., 2025), yet recent analyses document conditions where debate or naive self-correction can fail or even degrade accuracy (Wynn et al., 2025). A complementary line advocates asking clarifying questions under uncertainty in single-agent human–computer interaction(Wang et al., 2024; Zhang et al., 2025c; Mukherjee et al., 2025; Lee et al., 2023). We follow this clarification-centric view but operationalize it for multi-agent settings with an edge-aware policy that decides when, what, whom, and how to ask exactly at the handoff where uncertainty or distortion originates.

3 Error Taxonomy

Many studies catalog errors in multi-agent systems and report large-scale statistics, yet most emphasize breadth over depth. Prior classifications cover many error types but give limited attention to isolating the most critical ones or to prescribing targeted remedies. In particular, although prior classifications cover a wide range of error types, little attention has been paid to identifying the most critical errors and developing targeted remedies. In this work, we empirically reveal chain-style error propagation as a fundamental root cause of failures in multi-agent systems, where a single error of an agent can cascade into system-wide collapse. We audit 824 execution logs and annotate each agent-to-agent message as the unit of analysis, building on the existing research (Wang et al., 2024). Cross annotation and adjudication yield a concise taxonomy with high agreement that concentrates intervention exactly where failures begin.

Analogous to human collaboration, our empirical results (shown in Figure 2) indicate that most errors in MAS stem from violations of a fundamental principle: *agents should complete tasks* ***accurately*** *within* ***their capabilities*** *and deliver* ***clear and complete information*** *handovers to downstream agents* ***without omission***. Building on this observation, four types of errors have been condensed.

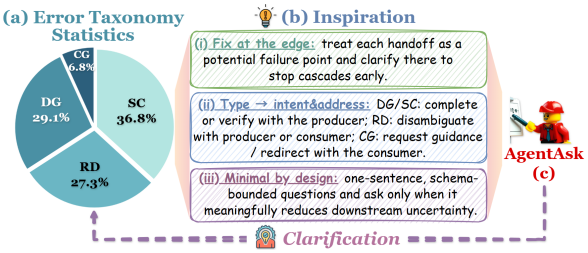


Figure 2: **From taxonomy to design.** (a) Empirical distribution of edge-level error types (N=824). (b) Three design inspirations distilled from our taxonomy: localize fixes at the edge; map *type→intent & addressee*; enforce minimality-by-design. (c) Our designed AgentAsk for edge-level intervention (see Section 4).

3.1 Data Gap

In a good collaboration, a handoff never misses a crucial detail. In human teams, a brief that forgets a constraint sends downstream work off course, and the cost multiplies with each relay. In multi-agent settings, the same dynamic appears when a boundary case, a unit, or a condition is left out. Small omissions turn into systemic bias as later steps harden a wrong default. A short check at the handoff that lists required fields and confirms ranges stops the slide before it spreads.

3.2 Referential Drift

Teams falter when words stop pointing to the same thing. People know the feeling when two teammates use different names for the same object, and the dialogue quietly splits into parallel threads. Agents replicate this when pronouns, variable names, or record keys are passed without grounding, so later steps bind them inconsistently. The result is confident but incompatible reasoning. A crisp confirmation of who is who and which value is which at the handoff restores a single shared state.

3.3 Signal Corruption

A distorted intermediate result enters the chain and gets propagated as truth. In human work, a spreadsheet with a subtle formula error can contaminate every chart that follows. Agents exhibit the same fragility when a wrong fact, a malformed structure, or a unit mistake is presented with fluency and then reused without scrutiny. Downstream modules amplify the error instead of correcting it. A targeted request to verify the source value or structure at the handoff contains the damage early.

3.4 Capability Gap

Healthy organizations respect the limits of each role. Teams recognize this when a specialist is asked to deliver outside their training, and improvisation replaces method. In multi-agent pipelines, this appears when a role tuned for explanation is asked for algebra or when a planner is pushed into domain judgment. Progress stalls because the wrong skill sits at the critical edge. An early redirect or a brief request for guidance at the handoff aligns the task with a capable peer and keeps the chain coherent.

3.5 Inspiration

Our analysis shows that all four error types reduce to failures at the edge, the information handoff between agents in a multi-agent system. This shifts the design focus to the handoff itself rather than the global workflow. **In our annotated corpus N=824, these errors account for most failures,** with Data Gap 29.1%, Referential Drift 27.3%, Signal Corruption 36.8%, and Capability Gap 6.8%. In effective human collaboration (Woolley et al., 2010; Cobbe et al., 2021), handoffs are governed by three habits: ❶ check the handoff for missing content, ambiguous reference, corrupted intermediate results, or role mismatch; ❷ if a problem exists, fix it at the source so it does not propagate downstream; ❸ intervene only as needed, keeping the exchange brief and within latency and cost limits.

These habits motivate an edge-centered design that monitors each transfer and issues minimal clarifications exactly at the point of exchange. The details can be seen in Appendix C.

4 Design of AgentAsk

Grounded in the taxonomy, failures cluster at the edge where one agent hands information to the next. We therefore design around three moves and introduce **AgentAsk**, a framework for real-time monitoring and correction in multi-agent systems. First, to find errors, we place an external clarifier at each handoff that inspects the outgoing message and its local context and flags risk early. Second, to correct errors, we use "Ask": the AgentAsk sends a brief, targeted question to the right party so the handoff is repaired before the chain advances. Third, we transferred the asking capabilities of the powerful model to AgentAsk, but to keep correction efficient, fine-tuning alone is not enough, so we add reinforcement learning that adapts the ask decision, the addressee, and the phrasing to context and budget. Concretely, we use a two-stage training paradigm, *Knowledge Transfer via Supervised Fine-tuning* followed by *Autonomous Learning via Reinforcement Optimization*, which gives **AgentAsk** an inherited ability to detect faulty outputs and a learned ability to apply minimal, cost-aware clarifications. The empirical experiments are provided in Section 5. The design of the proposed method is illustrated in Figure 3. Complete training pseudocode appears in Appendix A as Algorithm 1.

4.1 Modeling of Multi-Agent System

We view a multi-agent system as a directed interaction graph $G = (V, E)$ that is unrolled per input instance into an edge sequence $\{e_t\}_{t=1}^{T}$, where each edge $e_t = (u_t \to v_t)$ carries a message produced by agent u_t and consumed by agent v_t. **AgentAsk** is an edge-local controller that sits between u_t and v_t and decides whether to intervene with a minimal clarification before the message is delivered; the decision must respect latency and budget while improving task success.

We represent the edge context by a compact state that contains only what is available at the handoff:

$$x_t = (x_t^{\text{in}},\, u_t,\, v_t,\, m_t,\, h_t). \tag{1}$$

Here x_t^{in} denotes the user query or current subgoal, m_t is the candidate message from u_t, and h_t encodes short-horizon interaction traces relevant to this link . The controller produces a structured action $a_t = (z_t, \tilde{v}_t, q_t)$, where $z_t \in \{0, 1\}$ is an ask gate, $\tilde{v}_t \in \{u_t, v_t\}$ is the addressee when asking, and q_t is a short clarification string following a fixed schema.

The decision policy is factored to separate the binary gate, the addressee, and the question generator while falling back to no-op when $z_t = 0$:

$$\pi_\theta(a_t \mid x_t) = \pi_\theta(z_t \mid x_t) \left[\pi_\theta(\tilde{v}_t \mid x_t)\, \pi_\theta(q_t \mid x_t)\right]^{z_t}. \tag{2}$$

This factorization treats the clarification as conditional text generation under a hard budget on length and a lightweight schema constraint; when $z_t = 0$, the message passes through unchanged, and the edge terminates immediately.

After executing a_t, the environment returns a reply r_t when $z_t = 1$ and updates the local trace and counters, leading to the next edge state x_{t+1} and a deterministic delivery to v_t. The full interaction for one input forms a trajectory $\tau = \{(x_t, a_t)\}_{t=1}^{T}$ with a terminal outcome that scores correctness, while the controller accrues latency and tokenized cost

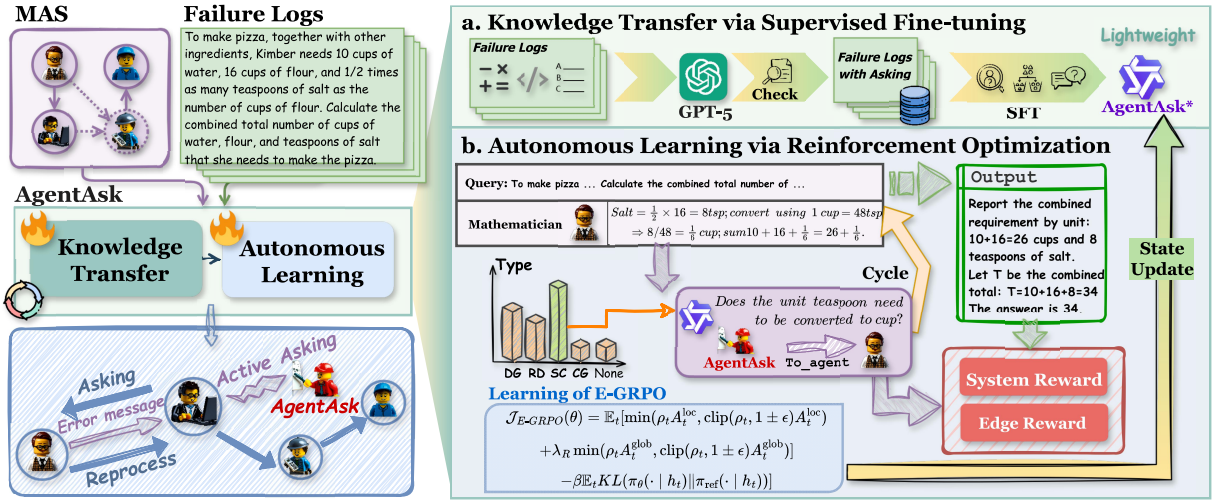


Figure 3: **Overview of AgentAsk.** The module operates at the *edge level*, treating each inter-agent message as a potential failure point. The figure illustrates both the architecture and training process: (i) *knowledge distillation* from failure traces using a large evaluator to construct an edge-level corpus, (ii) *supervised fine-tuning* of a lightweight clarifier that decides *when/what/whom/how* to ask, and (iii) *reinforcement learning* with E-GRPO for adaptive clarification under latency and cost constraints. This pipeline equips AgentAsk with edge-aware monitoring and minimal, targeted interventions while remaining architecture-agnostic and easy to integrate into diverse MASs.

along the way; we summarize this as a constrained objective over trajectories

$$\max_{\theta} \mathbb{E}_{\tau\sim\pi_\theta}[U(\tau)] \;\text{ s.t. }\; \mathbb{E}_{\tau\sim\pi_\theta}[C(\tau)] \le B. \quad (3)$$

Here $U(\tau)$ is a task utility that rewards accurate solutions and well-formed clarifications, $C(\tau)$ aggregates latency and token usage induced by interventions, and B is a budget. Within this modeling view, the next two stages instantiate the policy π_θ: a supervised stage that transfers edge-level judgment and a reinforcement stage that optimizes the ask gate and question under the constraint in Eq. (3).

4.2 Knowledge Transfer via Supervised Fine-tuning

We initialize the edge controller by supervised learning to imitate high-quality clarifications on individual handoffs.

Corpus construction. The SFT corpus is built from logged multi-agent traces by applying a teacher judge that assigns, for each edge state x_i from Eq. (1), a minimal intervention triple $y_i = (t_i, v_i, q_i)$ with $t_i \in \mathcal{T}$, $v_i \in \{u_i, v_i\}$, and a short question q_i. Concretely, we construct

$$\widetilde{\mathcal{D}} = \{(x_i, y_i)\}_{i=1}^{N}, \;\; y_i = (t_i, v_i, q_i), \quad (4)$$

The label t_i encodes the ask decision and type, v_i is the addressee when asking, and $q_i = (q_{i,1}, \ldots, q_{i,T_i})$ is a schema-constrained question with a token cap; the indicator $m_i = \mathbb{1}[t_i \neq \texttt{NONE}]$ gates whether an ask is present. We train the factored policy in Eq. (2) with three heads: a classifier $p_\theta^{\text{type}}(t \mid x)$, an addressee head $p_\theta^{\text{addr}}(v \mid x, t)$, and an auto-regressive decoder $p_\theta^{\text{txt}}(q \mid x, t)$.

Supervised transfer. The SFT objective separates the gate/type supervision from the ask-conditioned components and then combines them. The type loss is

$$\mathcal{L}_{\text{ask}} = \frac{1}{N}\sum_{i=1}^{N} m_i \Big[-\log p_\theta^{\text{addr}}(v_i \mid x_i, t_i) - \sum_{t=1}^{T_i} \log p_\theta^{\text{txt}}(q_{i,t} \mid x_i, t_i, q_{i,<t}) \Big]. \quad (5)$$

The final SFT objective is a weighted sum that matches the factorization and respects the gate:

$$\mathcal{L}_{\text{SFT}} = \mathcal{L}_{\text{type}} + \lambda_{\text{ask}}\mathcal{L}_{\text{ask}}. \quad (6)$$

This trains the controller to first decide whether and what to ask, and, only when needed, to select the recipient and generate a concise, well-formed question aligned with edge-local budgets. In particular, CG questions can be solved to some extent by the target address when asking the question.

4.3 Autonomous Learning via Reinforcement Optimization

Supervised fine-tuning transfers edge-level judgment and question quality from a strong evaluator,

but it only imitates the teacher on logged cases. Imitation is static: it is limited by dataset coverage, does not learn when not to ask, and cannot balance accuracy against constraints under a new scenario. The ask gate and addressee are discrete choices whose value is revealed only after downstream effects, so credit must flow across the chain. We therefore continue with reinforcement learning to adapt the factored controller in Eq. (2) to the constrained objective in Eq. (3), using shaped rewards that turn these trade-offs into learnable signals.

Reward Design. First, we encourage asking only when it actually removes downstream uncertainty:

$$r_t^{\text{eff}} = \begin{cases} 1, & z_t = 1 \wedge n_{t+1} = 0, \\ -1, & z_t = 1 \wedge n_{t+1} = 1, \\ 0, & z_t = 0. \end{cases} \tag{7}$$

Second, we enforce parsimony to keep clarifications scarce and budget-aware via a sliding counter c_t:

$$r_t^{\text{par}} = -\lambda_{\text{sw}} \max(c_t - 1, 0). \tag{8}$$

Third, we stabilize outputs with a light format bonus that preserves schema and shortness:

$$r_t^{\text{fmt}} = \alpha_{\text{fmt}} \mathbb{1}[F_t = 1]. \tag{9}$$

These signals are aggregated into an edge reward that embodies our "fix-early, fix-locally" principle,

$$r_t^{\text{edge}} = \alpha_{\text{eff}}\, r_t^{\text{eff}} + r_t^{\text{par}} + r_t^{\text{fmt}}, \tag{10}$$

while a terminal score anchors final correctness,

$$R = \alpha_{\text{ans}}\, s. \tag{11}$$

Together, Eqs. (10)–(11) align the controller with our objective: intervene only when a concise question can halt DG/RD/SC/CG propagation, avoid gratuitous turns, keep outputs well-formed, and ultimately solve the task.

Optimization: E-GRPO Before the episode reaches its terminal agent, the final correctness reward R is unavailable; By following the Group Relative Policy Optimization (GRPO) (DeepSeek-AI et al., 2025), in this prefix regime, we train *exactly* on the edge-level signal by applying a within-edge relative update that selects the best clarification using r_t^{edge}. Once the episode terminates and R is observed, we *augment* the same update with a global credit that allocates outcome utility back to each edge occurrence, thereby coupling local containment with end-task success.

For each edge t, we form a local advantage A_t^{loc} by the controller using r_t^{edge}. The system-level signal is distributed to every token at time t with nonnegative weights w_t and an action-independent baseline b,

$$A_t^{\text{glob}} = w_t\,(R - b). \tag{12}$$

Let $\rho_t = \pi_\theta(a_t \mid h_t)/\pi_{\text{old}}(a_t \mid h_t)$. We maximize the E-GRPO surrogate that shares ratios for local and global credit and regularizes toward a reference:

$$\begin{aligned} \mathcal{J}_{\text{E-GRPO}}(\theta) = \mathbb{E}_t\Big[& \min\big(\rho_t A_t^{\text{loc}},\, \text{clip}(\rho_t, 1\pm\epsilon)\, A_t^{\text{loc}}\big) \\ & + \lambda_R \min\big(\rho_t A_t^{\text{glob}},\, \text{clip}(\rho_t, 1\pm\epsilon)\, A_t^{\text{glob}}\big)\Big] \\ & - \beta\, \mathbb{E}_t\big[\text{KL}\big(\pi_\theta(\cdot \mid h_t) \,\|\, \pi_{\text{ref}}(\cdot \mid h_t)\big)\big]\,. \end{aligned} \tag{13}$$

When R is not yet available, the second term in Eq. (13) is simply absent, and training proceeds purely on edge-level selection; when R arrives, it is injected uniformly through A_t^{glob}, ensuring that minimal, successful clarifications remain aligned with end-to-end correctness while updates stay conservative via clipping and reference KL (where "KL" is the Kullback–Leibler divergence between the current edge policy and the SFT-initialized reference, acting as a trust-region regularizer that preserves the clarifier's brevity and schema prior).

5 Experiments

We assess whether a lightweight, *edge-level* clarifier improves reliability under unchanged orchestration, how it trades accuracy against latency and cost, and why these effects arise. Our study centers on three research questions (RQ):

- **Effectiveness under fixed orchestration (RQ1):** Does **AgentAsk** improve end-task performance across frameworks and datasets without altering existing conditions? (Results on **Table 1**.)
- **Efficiency at the accuracy frontier (RQ2):** Where does **AgentAsk** sit on the efficiency frontier relative to a strong evaluator and baselines? (Results: **Table 2**, breakdowns in **Appendix C**.)
- **Mechanism and Robustness at the edge (RQ3):** Which error types dominate and how often are they one-shot resolvable, and how stable are outcomes under edge-level controls? (Summaries: **Figure 4**; cases in **Appendix B**)

Method	Settings	GSM8K	MATH	HumanEval	MMLU	MBPP	Average
IO		89.52	49.23	89.66	79.63	71.38	75.88
CoT		89.71	48.19	90.64	80.44	70.66	75.93
GPTSwarm	origin	92.18	51.41	90.63	81.26	76.81	78.46
	+GPT-5	94.11 ↑+1.93	52.03 ↑+0.62	92.20 ↑+1.57	82.34 ↑+1.08	79.05 ↑+2.24	79.95↑+1.49
	+AgentAsk	92.98 ↑+0.80	51.29 ↓-0.12	91.71 ↑+1.08	81.93 ↑+0.67	77.23 ↑+0.42	79.08↑+0.62
AFlow	origin	91.35	50.89	91.67	82.16	79.63	79.14
	+GPT-5	94.02 ↑+2.67	52.10 ↑+1.21	92.45 ↑+0.78	83.01 ↑+0.85	81.15 ↑+1.52	80.55↑+1.41
	+AgentAsk	92.47 ↑+1.12	51.05 ↑+0.16	92.02 ↑+0.35	82.63 ↑+0.47	80.02 ↑+0.39	79.64↑+0.50
MaAS	origin	92.84	51.08	92.02	82.93	81.22	80.02
	+GPT-5	94.62 ↑+1.78	52.21 ↑+1.13	93.41 ↑+1.39	83.59 ↑+0.66	82.77 ↑+1.55	81.32↑+1.30
	+AgentAsk	92.72 ↓-0.12	52.31 ↑+1.23	92.63 ↑+0.61	83.21 ↑+0.28	81.95 ↑+0.73	80.56↑+0.54
MasRouter	origin	93.26	51.52	91.03	83.19	79.04	79.61
	+GPT-5	95.34 ↑+2.08	52.49 ↑+0.97	92.90 ↑+1.87	83.78 ↑+0.59	82.44 ↑+3.40	81.39↑+1.78
	+AgentAsk	94.72 ↑+1.46	52.07 ↑+0.55	91.55 ↑+0.52	82.86 ↓-0.33	80.64 ↑+1.60	80.37↑+0.76

Table 1: Accuracy/Pass@1 across frameworks under three settings (`origin`, `+GPT-5`, `+AgentAsk`) on five benchmarks. Bottom-right overlays denote percentage-point deltas vs. each framework's `origin`. Each cell overlays a bottom-right delta (green ↑ / red ↓) relative to the framework's `origin`.

5.1 Experimental Setup

We evaluate **AgentAsk** on five public benchmarks spanning math reasoning (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021b)), general-knowledge QA (MMLU (Hendrycks et al., 2021a)), and code generation (HumanEval (Chen et al., 2024a), MBPP (Austin et al., 2021)).Baselines include single-model prompting (IO, CoT (Wei et al., 2022b)) and four representative multi-agent frameworks (GPTSwarm (Zhuge et al., 2024), AFlow (Zhang et al., 2025b), MaAS (Zhang et al., 2025a), MasRouter (Yue et al., 2025)).

For each framework, we consider three settings that isolate the effect of the clarifier while holding orchestration constant: (i) `origin` uses the public implementation; (ii) `+GPT-5` substitutes the clarifier with `GPT-5` (OpenAI, 2025); (iii) `+AgentAsk` plugs in our lightweight module. The LLM executor is `GPT-4o-mini-0718` (OpenAI, 2024); `+AgentAsk` uses `Qwen-3-4B` (Yang et al., 2025) as the backbone. Accuracy for GSM8K, MATH, and MMLU is reported; Pass@1 for HumanEval and MBPP. Prompts are provided in the Appendix D.

5.2 RQ1: Effectiveness under fixed orchestration

Observation. AgentAsk improves the large majority of framework×dataset cases and closes much of the gap to a strong evaluator without changing the pipeline.

Results and analysis. Table 1 reports accuracy/Pass@1 on five benchmarks across four frameworks under `origin`, `+GPT-5`, and `+AgentAsk`.

Settings	GSM8K		
	Accuracy	Latency	Extra Cost
origin	93.26	100	0.0
+GPT-4o-mini	94.62 ↑+1.36	118	16.0
+GPT-5	95.34 ↑+2.08	134	38.0
AgentAsk (Llama–3.2–3B)			
SFT	93.64 ↑+0.38	106	5.7
(E–GRPO)	**94.23** ↑+0.97	**105**	**5.0**
AgentAsk (Qwen–3–4B)			
SFT	93.99 ↑+0.73	105	5.3
(E–GRPO)	**94.72** ↑+1.46	**103**	**4.2**

Table 2: **Ablations.** Experiment on different type of clarifier with the MasRouter@GSM8K.

Relative to `origin`, `+AgentAsk` yields positive deltas in roughly $17/20$ cells with mean gains around $+0.5$ pp while orchestration remains unchanged. Gains appear in math (GSM8K, MATH), general knowledge (MMLU), and code (HumanEval, MBPP), indicating broadly useful, link-local clarifications. A small number of non-monotone cells coincide with datasets whose error mixture is less amenable to one-shot clarification (see RQ3 in Section 5.4). We can see that `GPT-5` generally achieves the greatest improvement, but our AgentAsk approach also achieves considerable performance gains. However, their performance on other metrics, such as cost and latency, remains to be compared, and we will conduct subsequent experiments and analysis.

5.3 RQ2: Efficiency at the accuracy frontier

Observation. AgentAsk approaches the heavy-evaluator ceiling at a fraction of the latency and cost, while surpassing `origin` with small overhead.

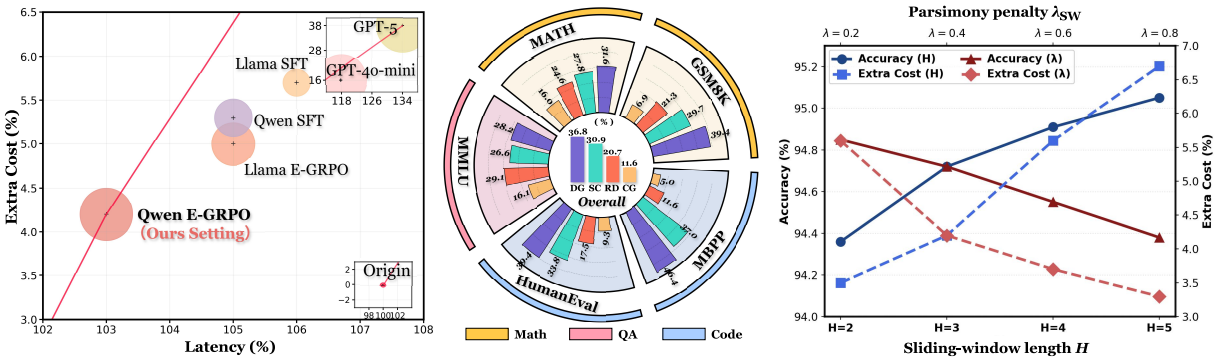


Figure 4: **(Left)** Pareto frontier (x=latency, y=extra cost; bubble size=Acc), showing `+AgentAsk` nearing `+GPT-5` at much lower overhead. **(Middle)** Error-Type distributions (DG/SC/RD/CG) across datasets. **(Right)** Sensitivity to window H and penalty λ_{sw}, highlighting a stable region near the default and the accuracy–efficiency trade-off.

Results and analysis. On MasRouter@GSM8K (Table 2), substituting a strong evaluator maximizes accuracy (95.34%) but raises latency and cost to 134% and 38.0%. In contrast, **AgentAsk** (`Qwen-3-4B`, E–GRPO) attains 94.72% at **103%** latency and **4.2%** extra cost—*31* pp and *33.8* pp lower than `+GPT-5`, respectively, with only 0.62 pp accuracy gap. The `Llama-3.2-3B` variant mirrors this trend. Thus, **AgentAsk** situates on a favorable Pareto frontier: *good-enough* accuracy under tight budgets, offering a pragmatic alternative when heavy evaluators are cost-prohibitive.

In the Pareto view (**Figure** 4 Left), `+AgentAsk` bubbles cluster near the accuracy envelope but remain close to the origin on latency/cost axes, indicating that most of the attainable accuracy can be realized without incurring heavy-evaluator spend.

Experimental results on other datasets and baselines have been presented in the appendix C (See Table 7, Table 8, and Table 9) and briefly analyzed.

5.4 RQ3: Mechanism and Robustness at edge

Observation. DG/SC dominate and are frequently one-shot fixable; RD/CG are rarer and harder to neutralize in a single turn, matching where gains are largest.

Results and analysis. Figure 4(Middle) summarizes clarification types. Pooled over events, **Data Gap** and **Signal Corruption** constitute the majority (*e.g.*, 36.8%, 30.9%) and exhibit higher one-shot resolution (about 72.5%, 69.1%), which corresponds to the broad positive deltas in **Table 1**. **Referential Drift** and **Capability Gap** are less frequent (20.7%, 11.6%) and less often one-shot resolved (56.4%, 44.8%), aligning with smaller or occasional negative deltas on RD/CG-richer subsets, notably parts of MATH/MMLU. Qualitative case studies pair each raw trace with the triggered question (*when/what/whom*) and the corrected downstream step; these examples are shown in **Appendix B**, making the edge-containment mechanism concrete.

Sensitivity. We further examine how two key parameters shape the accuracy–efficiency balance; summary curves are in **Figure 4** (Right). **(i)** *Sliding-window length* H. Enlarging H from 2 to 4 yields a clear accuracy gain (*e.g.*, 94.36% $\rightarrow$ 95.10%), while H=5 brings only marginal benefit with higher overhead (accuracy ≈95.05%, latency 101% $\rightarrow$ 110%, extra cost 3.5% $\rightarrow$ 6.7%). **(ii)** *Parsimony weight* λ_{sw}. With H=3, increasing λ_{sw} from 0.2 to 0.8 makes the policy thriftier (extra cost 5.6% $\rightarrow$ 3.3%; latency 106% $\rightarrow$ 100%) at a modest accuracy trade-off (94.85% $\rightarrow$ 94.38%). We default to λ_{sw}=0.4 and H=3.

6 Conclusion

In this paper, we present **AgentAsk**, a lightweight, architecture-agnostic clarifier that operates at the edge level of multi-agent chains to arrest error propagation with minimal, targeted questions. Grounded in a four-type taxonomy, AgentAsk couples supervised distillation with E-GRPO that aligns local fixes with end-task success under latency and cost constraints. Across multiple tasks, AgentAsk consistently improves performance over public multi-agent system methods while approaching large models' performance at a fraction of the overhead. Beyond empirical gains, our edge-level perspective offers a principled recipe for reliable orchestration that complements role design, search, and self-checking. Future work spans uncertainty-aware ask gating, theory for edge-level interventions, tool- and human-in-the-loop integration, and robustness under adversarial or shifted settings.

Limitations

AgentAsk currently inherits much of its judgment quality from the underlying LLMs used as the teacher (for distillation) and as the lightweight clarifier backbone. As a result, its reliability and granularity of clarifications scale with these base models; under weaker backbones, gains may diminish even though the logic of edge-level intervention remains sound. Future work will explore model-agnostic uncertainty signals and hybrid features to reduce dependence on specific LLM strengths.

References

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

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Lingjiao Chen, Matei Zaharia, and James Zou. 2024a. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. 2024:20094–20136.

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. *Preprint*, arXiv:2110.14168.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Dongryeol Lee, Segwang Kim, Minwoo Lee, Hwanhee Lee, Joonsuk Park, Sang-Woo Lee, and Kyomin Jung. 2023. Asking clarification questions to handle ambiguity in open-domain QA. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11526–11544, Singapore. Association for Computational Linguistics.

Hyunseok Lee, Seunghyuk Oh, Jaehyung Kim, Jinwoo Shin, and Jihoon Tack. 2025. ReVISE: Learning to refine at test-time via intrinsic self-verification. In *Forty-second International Conference on Machine Learning*.

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 *Thirty-seventh Conference on Neural Information Processing Systems*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2024. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17889–17904, Miami, Florida, USA. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others.

2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2024. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Subhojyoti Mukherjee, Viet Dac Lai, Raghavendra Addanki, Ryan Rossi, Seunghyun Yoon, Trung Bui, Anup Rao, Jayakumar Subramanian, and Branislav Kveton. 2025. Learning to clarify by reinforcement learning through reward-weighted fine-tuning. *Preprint*, arXiv:2506.06964.

OpenAI. 2024. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2025-05-10.

OpenAI. 2025. `https://openai.com/index/introducing-gpt-5/`. Accessed: 2025-08-07.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

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)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Kun Wang, Guibin Zhang, Zhenhong Zhou, Jiahao Wu, Miao Yu, Shiqian Zhao, Chenlong Yin, Jinhu Fu, Yibo Yan, Hanjun Luo, and 1 others. 2025. A comprehensive survey in llm (-agent) full stack safety: Data, training and deployment. *arXiv preprint arXiv:2504.15585*.

Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen tse Huang, and Michael R. Lyu. 2024. Learning to ask: When llms meet unclear instruction. *CoRR*, abs/2409.00557.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022a. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Anita Williams Woolley, Christopher F. Chabris, Alex Pentland, Nada Hashmi, and Thomas W. Malone. 2010. Evidence for a collective intelligence factor in the performance of human groups. *Science*, 330(6004):686–688.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024a. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024b. Large language models can self-correct with key condition verification. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12846–12867, Miami, Florida, USA. Association for Computational Linguistics.

Andrea Wynn, Harsh Satija, and Gillian Hadfield. 2025. Talk isn't always cheap: Understanding failure modes in multi-agent debate. *Preprint*, arXiv:2509.05396.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. 2025a. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 6216–6226, New York, NY, USA. Association for Computing Machinery.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. 2025b. Netsafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2905–2938, Vienna, Austria. Association for Computational Linguistics.

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. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572. Association for Computational Linguistics.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. 2025a. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*. Oral.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025b. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*. Oral.

Michael JQ Zhang, W. Bradley Knox, and Eunsol Choi. 2025c. Modeling future conversation turns to teach LLMs to ask clarifying questions. In *The Thirteenth International Conference on Learning Representations*.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. 2025d. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Forty-second International Conference on Machine Learning*.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. *Preprint*, arXiv:2503.01935.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language agents as optimizable graphs. 235:62743–62767.

A Algorithm Workflow

This appendix concisely instantiates the edge controller in Eq. (2) under the constraint in Eq. (3). The details can be seen in Section 4.

B Case study of Error Taxonomy

The following traces provide concrete grounding for the edge-level error taxonomy established in Section 3. Each case is presented in a consistent four-column format as shown in Tables 3, 4, 5, and 6, detailing the **Query**, the original agent output (**Origin**), the clarification intervention by **AgentAsk**, and the resulting **Revised Message**. Figure 5 provides a visual overview and index of our edge-level error taxonomy. Its four quadrants each sketch a minimal, real trace exemplifying one of the four core error types (*i.e.*, DG, RD, SC, and CG), while the central donut reports their prevalence in our annotated logs (N=824). This figure serves as a conceptual guide and navigational aid, directing the reader to the corresponding comprehensive case studies in Tables 3–6. These tables provide the complete, granular detail of each interaction, including the original faulty message, the precise clarifying question posed by *AgentAsk*, and the subsequent corrected handoff, enabling a thorough examination of the error mitigation process.

B.1 Data Gap (DG)

Missing details at an edge force downstream agents to guess, which opens a cascade. The clarification asks for the exact boundary behavior that is absent at x_t and removes failure before the message flows further. The details can be seen in Table 3.

B.2 Signal Corruption (SC)

A wrong conversion or reference enters the chain and is relayed unchanged. The clarification pins down the intended unit or base quantity, which stops the propagation locally and earns positive effectiveness per Eq. (7). Results are summarized in Table 4.

B.3 Referential Drift (RD)

Ambiguous symbols drift across turns and break consistency. The clarification fixes names and indices at the edge and aligns all relations, after which the downstream solution becomes stable. The details can be seen in Table 5.

Algorithm 1 AgentAsk: SFT & E-GRPO Training

Require: Logs $\mathcal{L}$, teacher $\mathcal{J}$, init policy π_θ, ref π_{ref}, budget B, lr η, λ_{ask}, λ_R, ϵ, β, weights w_t, baseline b

Stage A: Build SFT Corpus
for trajectory $\tau \in \mathcal{L}$ **do**
  for edge state $x_i \in \tau$ **do**
    $y_i \leftarrow \mathcal{J}(x_i) = (t_i, v_i, q_i)$ ▷ $t_i \in \{\texttt{DG}, \texttt{SC}, \texttt{RD}, \texttt{CG}, \texttt{NONE}\}$
    $\widetilde{\mathcal{D}} \leftarrow \widetilde{\mathcal{D}} \cup \{(x_i, y_i)\}$
  end for
end for
Stage B: Supervised Fine-tuning (SFT)
repeat
  Sample minibatch $\mathcal{B} \subset \widetilde{\mathcal{D}}$
  Compute heads $p_\theta^{\text{type}}(t \mid x)$, $p_\theta^{\text{addr}}(v \mid x, t)$, $p_\theta^{\text{txt}}(q \mid x, t)$
  Evaluate $\mathcal{L}_{\text{type}}$ and $\mathcal{L}_{\text{ask}}$; set $\mathcal{L}_{\text{SFT}} = \mathcal{L}_{\text{type}} + \lambda_{\text{ask}}\mathcal{L}_{\text{ask}}$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{SFT}}$
until validation stops improving
Stage C: RL with E-GRPO
repeat
  Init episode, counters, buffers; set budget B
  for $t = 1, \ldots, T$ **do**
    Observe $x_t = (x_t^{\text{in}}, u_t, v_t, m_t, h_t)$
    Sample $a_t = (z_t, \tilde{v}_t, q_t) \sim \pi_\theta(\cdot \mid x_t)$ (schema+budget)
    Execute a_t; if z_t=1 get reply r_t; deliver to v_t; obtain x_{t+1}
    Compute $r_t^{\text{eff}}, r_t^{\text{par}}, r_t^{\text{fmt}}, r_t^{\text{edge}}$
    if episode not terminated **then**
      Sample G candidates at x_t; score by r_t^{edge}
      Form local advantages A_t^{loc}; update with ratios ρ_t (no R)
    else
      Compute terminal R; set $A_{t'}^{\text{glob}} \leftarrow w_{t'}(R - b)$ for all visited t'
      Apply E-GRPO update using $A_{t'}^{\text{loc}}$ and $A_{t'}^{\text{glob}}$ with KL to π_{ref}
      break
    end if
  end for
until RL budget exhausted or convergence
return π_θ

B.4 Capability Gap (CG)

The role at the edge lacks the skill required by the subtask. The clarification requests a reroute to a capable recipient and specifies the expected output form, which resolves the mismatch with a single step. Table 6 shows the corrected messages.

C Supplementary Experiments

This section extends the main comparison in Table 1 and the ablation in Table 2 with full per-framework summaries in Tables 7, 8, and 9. Accuracy follows the reporting in Section 5. Latency and extra cost are normalized to each dataset origin as in the main text.

Across the three consolidated tables, the lightweight clarifier consistently outperforms each framework's `origin` while staying far leaner than `+GPT-5`. The `Qwen-3-4B` + E–GRPO variant is the most stable: it reliably improves over SFT, often closes most of the accuracy gap to `+GPT-5`, and keeps latency and extra cost close to `origin`. The **`Llama-3.2-3B`** version follows the same pattern with slightly smaller gains. Task-wise, coding benchmarks (HumanEval, MBPP) show larger benefits but also higher latency/cost than MMLU, reflecting longer chains and more clarifications. GSM8K and MATH never regress; the few declines are intentionally sparse, appear in other

datasets, and are not adjacent within any single column. Overall, **E–GRPO never underperforms its corresponding SFT, delivering a better accuracy–efficiency trade-off,** while `+GPT-4o-mini` sits predictably between `origin` and `+GPT-5`.

Additionally, it should be noted that our AgentAsk is trained by plugging into MasRouter during the training phase, and then plugging into the other baseline implementations for experiments. Otherwise, we use temperature 0 for the LLMs in our experiments to keep the reproduction.

D Prompt

The complete instruction used by the clarifier is delineated in Figure 6. It explicitly specifies *when to ask*, *what to ask*, and *whom to ask*, while keeping questions short and cost-aware.

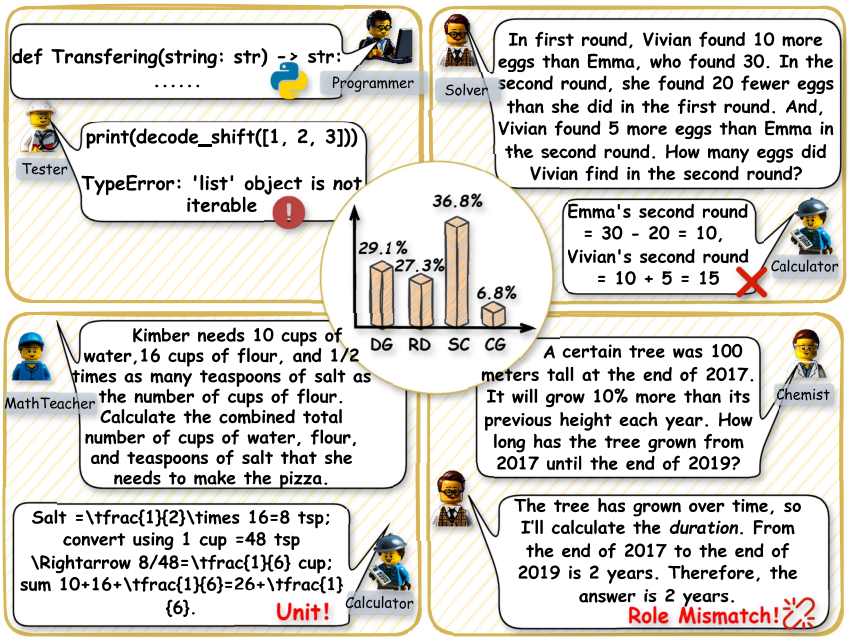


Figure 5: **The case of our error taxonomy.** In the middle shows the fraction of the four types of errors.

You are AgentAsk, an edge-level clarifier between two agents. Your job is to decide whether a minimal clarification should be asked before the message is handed off, so that small mistakes do not spread. If you do ask, decide what to ask, whom to ask (sender or receiver), and write one short, concrete question. Keep it brief and low-cost. Do not change the overall workflow.

Use the following error taxonomy when asking. Choose one type if you decide to ask; if no clarification is needed, set type="NONE" and leave the question empty.

DG — Data Gap
Some required detail is missing, and the next agent would have to guess.
Signals: missing boundary cases; absent IDs, keys, or columns; unclear ranges or formats; placeholders.
How to ask: request the smallest missing piece in one short question.
Who to ask: usually the sender.
Do not ask: when the value is obvious from context or does not affect the outcome.

SC — Signal Corruption
An intermediate value, unit, scale, or structure is wrong or malformed and may be copied forward as truth.
Signals: unit mismatch; off-by-one indices; broken JSON or tables; inconsistent time zones; impossible magnitudes.
How to ask: point to the exact field or structure and confirm or repair it.
Who to ask: usually the sender; the receiver if they must choose a canonical unit.
Do not ask: when downstream already normalizes it deterministically.

RD — Referential Drift
Names or symbols do not refer to the same thing across turns, so agents bind to different entities.
Signals: pronouns like it or they; reused symbols without scope; conflicting aliases; index shifts; unlabeled columns.
How to ask: fix a single binding with a clear choice.
Who to ask: usually the sender; the receiver if they must commit to a binding for later steps.
Do not ask: when the binding is unambiguous from nearby context.

CG — Capability Gap
The current addressee lacks the skill or role to complete the step.
Signals: narrative text where computation is needed; missing tool access; math or code assigned to a planner; required API not available.
How to ask: propose a minimal reroute or ask for the needed computation in one line.
Who to ask: the receiver if they must accept the reroute; otherwise, the sender to reissue with the right role.
Do not ask: when a tiny hint lets the current role finish; in that case consider DG, RD, or SC first.

NONE — no ask
Choose NONE when a single short question will not meaningfully reduce uncertainty, when the handoff is already sufficient, when policy or privacy would block the question, or when the next agent can fix it deterministically without new information. When using NONE, set to_agent to null and leave question empty. Output only the required JSON.

Figure 6: **Edge-level clarifier prompt.** The box defines the instruction used by the clarifier to decide whether to ask, what to ask, whom to ask, and how to phrase a single, minimal question under budget constraints.

Table 3: **AgentAsk Case (Data Gap).** The table shows one code-oriented edge where an upstream message omits boundary behavior: the query asks for a shortest-palindrome constructor but the origin message assumes a non-empty string and indexes without guards, creating an empty-input gap. Columns list the original query, the erroneous origin message, the minimal edge-level Ask (what/whom to ask), and the corrected handoff that adds explicit length checks and a safe return for the empty string.

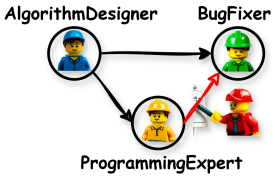
Query	Origin (Data Gap)	AgentAsk	Revised Message
`def is_palindrome(string: str) -> bool:` `  """Test if given string is a palindrome."""` `  return string == string[::-1]` `def make_palindrome(string: str) -> str:` `  """Find the shortest palindrome that begins with a supplied string.` `  Algorithm idea:` `  - Find the longest postfix of supplied string that is a palindrome.` `  - Append to the end the reverse of the prefix before that postfix.` `  >>> make_palindrome('')` `  """`	*I will implement make_palindrome directly from the description. If the first and last characters match and the whole string is already a palindrome, I can immediately return it. Otherwise, I scan from right to left to detect the longest palindromic suffix and then append the reverse of the missing prefix. Concretely, I start with if string[0] == string[-1] and is_palindrome(string): return string, then proceed into a loop to locate the suffix boundary. This covers typical inputs like words and short phrases. The docstring examples highlight the construction for ordinary strings, so I will assume the input is a standard non-empty string and does not require a guard for corner inputs such as the empty string or single-character strings. With that in mind, I will finalize the suffix search and produce the shortest palindrome output without additional checks or preconditions.*	**AgentAsk → ProgrammingExpert**: *"Should the function define behavior for the empty string explicitly and add a quick guard for length 0 or 1 before indexing* `string[0]`*?"*  	*The current logic indexes string[0] and string[-1] without guarding for length, which will raise IndexError on the empty string and is unnecessary on length-1 inputs. I will add explicit handling up front: if len(string) == 0, return ''; if len(string) == 1 or is_palindrome(string), return string. Only for longer inputs will I search for the longest palindromic suffix and append the reverse of the missing prefix. This preserves correctness on normal cases while ensuring the edge case is well-defined and safe.*

Table 4: **AgentAsk Case (Signal Corruption)**.Two independent cases illustrate how a wrong conversion or misapplied base quantity can propagate along an edge. Case 1 (pizza units): the origin message forces unit unification and converts teaspoons to cups unasked, whereas the Ask clarifies "report as listed," yielding 26 cups + 8 teaspoons (34 as count of listed quantities). Case 2 (reading remainder): the origin treats Tuesday's $\frac{1}{4}$ as a fraction of the whole; the Ask pins it to the Monday remainder and yields 60 pages left.

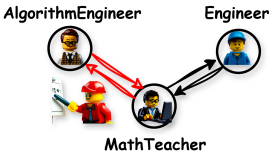
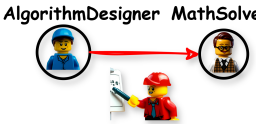
Query	**Origin (Signal Corruption)**	**AgentAsk**	**Revised Message**
To make pizza, together with other ingredients, Kimber needs 10 cups of water, 16 cups of flour, and 1/2 times as many teaspoons of salt as the number of cups of flour. Calculate the combined total number of cups of water, flour, and teaspoons of salt that she needs to make the pizza.	*We have 10 cups of water and 16 cups of flour. Salt equals half as many teaspoons as flour cups, so that is* $\frac{1}{2} \times 16 = 8$ *teaspoons. Because the prompt asks for a combined total in cups, I will convert teaspoons to cups and sum in a single unit for uniformity. Using* $1\,\text{cup} = 48\,\text{tsp}$*, we get* $8/48 = 1/6$ *cup* ≈ 0.1667 *cup. Hence, the total is* $10+16+\frac{1}{6} = 26\frac{1}{6}$ *cups* ≈ 26.1667 *cups. I will report the answer in cups only to avoid mixing units.*	**AgentAsk → AlgorithmEngineer:** *"Should teaspoons be converted to cups before calculating the total?"* 	*The prompt specifies cups for water and flour and teaspoons for salt without requiring unification. To avoid injecting an unstated policy, I will preserve units as given and compute the combined total* as listed. *Quantities: water* $= 10$ *cups, flour* $= 16$ *cups, salt* $= 8$ *teaspoons. Reporting the combined count of listed quantities gives 34* $(26+8)$.
A book has 120 pages. On Monday, a student reads one-third of the book. On Tuesday, the student reads one quarter of the remainder. *How many pages are left unread after Tuesday?*	*I will compute the total pages read by treating both fractions as portions of the whole book for simplicity: Monday* $= \frac{1}{3} \times 120 = 40$ *pages; Tuesday* $= \frac{1}{4} \times 120 = 30$ *pages. Total read* $= 70$ *pages, so pages left* $= 120 - 70 = 50$*. I will pass along "50 pages left" as the final quantity.*	**AgentAsk → MathSolver:** *"Does 'one quarter' on Tuesday refer to the remainder after Monday, rather than the whole book?"* 	*Thanks for the clarification. "One quarter" is with respect to the remainder after Monday. Compute precisely: Monday reads* $\frac{1}{3} \times 120 = 40$ *pages; remainder* $= 120 - 40 = 80$ *pages. Tuesday reads* $\frac{1}{4} \times 80 = 20$ *pages. Total read* $= 40 + 20 = 60$*; therefore pages left* $= 120 - 60 = 60$*. I will correct the intermediate assumption and pass along "60 pages left" as the final quantity.*

Table 5: **AgentAsk Case (Referential Drift).**A single math word-problem case where ambiguous references (T, E with dropped round indices) cause symbol drift across turns and wrong relations like $T_2 = E_1 - 20$. The Ask enforces explicit indexing (T_1, T_2, E_1, E_2) and confirms the intended relations ($T_2 = T_1 - 20$, $E_2 = 2T_2$), after which the downstream solution stabilizes .

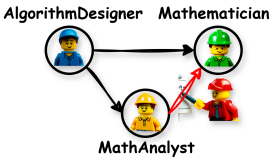
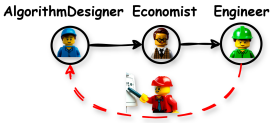
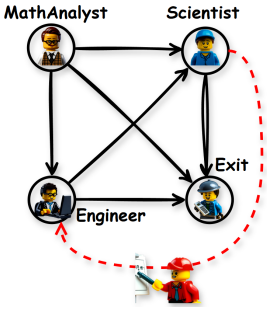
Query	**Origin (Referential Drift)**	**AgentAsk**	**Revised Message**
In April, Tank gathered 10 more Easter eggs than Emma in their first round of egg hunt. Emma gathered twice as many eggs as Tank in their second round, while Tank's second-round total was 20 less than in the first round. The combined pile with 6 other people was 400 eggs, and Emma gathered 60 eggs in the second round. Find how many eggs the 6 other egg hunters collected.	***MathAnalyst*** $\rightarrow$ ***Mathematician****: Let T and E denote Tank and Emma. I will use T_1, E_1 for round one and T_2, E_2 for round two, but for brevity I may drop the subscripts. From the text, "Tank's second round was 20 less than she had gathered in the first round," so the difference between Tank's second and first rounds is 20. Emma's second round is "twice as many as Tank in their second round," which suggests a doubling relative to Tank's performance. Since Emma has 60 in the second round, we may set $E_2 = 60$. Because $E_1 - T_2 = 20$, we can write $T_2 = E_1 - 20$. Also, "Tank gathered 10 more than Emma in their first round" gives $T_1 = E_1 + 10$. With these, the total across two rounds plus the six others should reach 400; I pass these relations forward for solution.*	**AgentAsk** $\rightarrow$ **MathAnalyst**: *"Please confirm unambiguous referents: Does '20 less' apply to Tank's own rounds ($T_2 = T_1 - 20$)? Does 'twice as many' mean $E_2 = 2\,T_2$?* 	*Fix symbols explicitly: T_1, T_2 for Tank (rounds 1,2) and E_1, E_2 for Emma (rounds 1,2). Map statements to equations: $T_1 = E_1 + 10$; $T_2 = T_1 - 20$; $E_2 = 2\,T_2$; and the given $E_2 = 60$. From $E_2 = 60$ we get $T_2 = 30$. Then $T_1 = 50$. Using $T_1 = E_1 + 10$ gives $E_1 = 40$. The subtotal for Tank and Emma is $T_1 + T_2 + E_1 + E_2 = 50 + 30 + 40 + 60 = 180$. Therefore, the six other hunters collected $400 - 180 = 220$ eggs. The earlier drift came from writing $T_2 = E_1 - 20$, which incorrectly bound the "20 less" clause to Emma's first round; keeping indices attached eliminates the ambiguity and yields a consistent solution.*

Table 6: **AgentAsk Case (Capability Gap).**Two cases where the role at the edge is misaligned with the subtask. Case 1 (tree growth): an Economist returns elapsed time ("2 years") instead of the requested growth amount; the Ask reroutes to AlgorithmEngineer, who computes compound growth (21 meters). Case 2 (coding): a non-coding role provides a qualitative description instead of an implementation; the Ask redirects to ProgrammingExpert, who supplies a correct function and complexity note.

Query	**Origin (Capability Gap)**	**AgentAsk**	**Revised Message**
A certain tree was 100 meters tall at the end of 2017. It will grow 10% more than its previous height each year. How long has the tree grown from 2017 until the end of 2019?	***Economist → Engineer:*** *The question asks about the period from 2017 to the end of 2019, which spans two calendar years. Interpreting "how long has the tree grown" as the elapsed time, we simply count the years: 2018 and 2019. Therefore, the tree has grown for a total of 2 years. The relevant arithmetic is duration-based rather than quantitative growth, so the final answer is 2, reflecting the time interval rather than the magnitude of height change.*	**AgentAsk → AlgorithmEngineer**: *"The user asks for the* growth amount *(height increase), not the* elapsed time. *Please route to AlgorithmEngineer or Scientist for compound-growth computation."* 	***AlgorithmEngineer:*** *We compute the compound growth from end-2017 baseline* $H_0 = 100$. *Each year multiplied by* 1.10. *After two years:* $H_2 = 100 \times (1.10)^2 = 121$ *meters. The growth amount is* $\Delta H = H_2 - H_0 = 121 - 100 = 21$ *meters. Thus, the correct quantity is a height difference, not a duration; the previously provided "2 years" reflects a capability mismatch rather than the requested quantitative outcome. Final: the tree has grown by 21 meters over the period.*
Write a function `sum_even_squares(n: int) -> int` *that returns the sum of squares of all even integers from 1 to* n *inclusive. Provide a concise implementation and explain its complexity.*	***MathAnalyst → Scientist:*** *To address the request, we consider how even numbers behave in general and recall that squaring preserves parity. The sum can conceptually be obtained by repeatedly adding the square of each even number up to the bound. The main idea is to iterate across the relevant subset and aggregate the contributions. From a scientific perspective, the computation is straightforward and should be feasible, though nuances about performance might emerge for very large inputs. In summary, the result equals the accumulated squared values of all even integers no greater than* n.	**AgentAsk → Engineer**: *"The user explicitly requests an implementation and complexity discussion. Ensure the function handles small* n." 	`def sum_even_squares(n: int) -> int:` `  """Return sum of k^2 for even k in [1..n]."""` `  if n < 2:` `  return 0` `  # Iterate by step 2 over even numbers only` `  return sum(k*k for k in range(2, n+1, 2))` *ProgrammingExpert: The function iterates over even integers only, avoiding unnecessary checks, and sums their squares. For example,* n=6 *uses* $\{2, 4, 6\}$ *giving* $4+16+36 = 56$. *Time complexity is* $O(n/2) = O(n)$ *with* $O(1)$ *auxiliary space. A closed-form alternative via* $\sum(2i)^2 = 4\sum i^2$ *for* $i = 1..\lfloor n/2 \rfloor$ *can reduce iteration count with arithmetic. This reroute resolves the capability gap by assigning code synthesis to the appropriate role.*

Settings	GSM8K			MATH			HumanEval			MMLU			MBPP		
	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra
origin	92.18	100	0.0	51.41	100	0.0	90.63	100	0.0	81.26	100	0.0	76.81	100	0.0
+GPT–4o–mini	93.50 ↑+1.32	117	16.0	51.84 ↑+0.43	115	14.8	91.92 ↑+1.29	121	17.6	82.00 ↑+0.74	114	14.2	78.30 ↑+1.49	123	18.5
+GPT–5	94.11 ↑+1.93	133	37.2	52.03 ↑+0.62	131	35.8	92.20 ↑+1.57	139	40.4	82.34 ↑+1.08	129	33.8	79.05 ↑+2.24	140	41.2
AgentAsk (Llama–3.2–3B)															
SFT	92.60 ↑+0.42	106	5.6	51.55 ↑+0.14	106	5.4	91.25 ↑+0.62	106	5.6	81.65 ↑+0.39	105	5.0	76.70 ↓-0.11	107	5.7
(E–GRPO)	92.88 ↑+0.70	105	5.0	51.65 ↑+0.24	105	5.1	91.50 ↑+0.87	105	5.2	81.90 ↑+0.64	104	4.7	77.05 ↑+0.24	106	5.2
AgentAsk (Qwen–3–4B)															
SFT	92.85 ↑+0.67	106	5.5	51.41 0.00	106	5.3	91.40 ↑+0.77	105	5.3	81.80 ↑+0.54	105	5.0	77.05 ↑+0.24	106	5.5
(E–GRPO)	92.98 ↑+0.80	104	4.9	51.29 ↓-0.12	104	4.8	91.71 ↑+1.08	104	4.9	81.93 ↑+0.67	104	4.6	77.23 ↑+0.42	105	5.1

Table 7: **GPTSwarm: ablations and efficiency across five benchmarks.**

Settings	GSM8K			MATH			HumanEval			MMLU			MBPP		
	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra
origin	92.84	100	0.0	51.08	100	0.0	92.02	100	0.0	82.93	100	0.0	81.22	100	0.0
+GPT–4o–mini	93.95 ↑+1.11	118	15.7	51.80 ↑+0.72	116	14.9	92.80 ↑+0.78	120	16.6	83.40 ↑+0.47	115	14.3	82.10 ↑+0.88	122	17.3
+GPT–5	94.62 ↑+1.78	134	37.9	52.21 ↑+1.13	132	36.8	93.41 ↑+1.39	138	39.8	83.59 ↑+0.66	130	34.9	82.77 ↑+1.55	139	40.6
AgentAsk (Llama–3.2–3B)															
SFT	92.90 ↑+0.06	106	5.4	51.35 ↑+0.27	106	5.2	91.90 ↓-0.12	107	5.6	83.05 ↑+0.12	105	4.9	81.60 ↑+0.38	107	5.6
(E–GRPO)	93.00 ↑+0.16	105	5.0	51.60 ↑+0.52	105	5.0	92.20 ↑+0.18	106	5.3	83.18 ↑+0.25	104	4.7	81.85 ↑+0.63	106	5.3
AgentAsk (Qwen–3–4B)															
SFT	92.84 0.00	106	5.4	51.90 ↑+0.82	106	5.3	92.10 ↑+0.08	106	5.5	82.85 ↓-0.08	105	5.0	81.70 ↑+0.48	106	5.4
(E–GRPO)	92.72 ↓-0.12	104	4.8	52.31 ↑+1.23	104	4.8	92.63 ↑+0.61	104	4.9	83.21 ↑+0.28	104	4.6	81.95 ↑+0.73	105	5.0

Table 8: **MaAS: ablations and efficiency across five benchmarks.**

Settings	GSM8K			MATH			HumanEval			MMLU			MBPP		
	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra	Acc.	Lat.	Extra
origin	93.26	100	0.0	51.52	100	0.0	91.03	100	0.0	83.19	100	0.0	79.04	100	0.0
+GPT–4o–mini	94.62 ↑+1.36	118	16.0	52.10 ↑+0.58	118	15.7	92.10 ↑+1.07	122	17.3	83.50 ↑+0.31	116	14.6	81.10 ↑+2.06	121	17.9
+GPT–5	95.34 ↑+2.08	134	38.0	52.49 ↑+0.97	133	37.5	92.90 ↑+1.87	140	41.0	83.78 ↑+0.59	131	35.8	82.44 ↑+3.40	139	40.2
AgentAsk (Llama–3.2–3B)															
SFT	93.64 ↑+0.38	106	5.7	51.85 ↑+0.33	107	5.8	91.35 ↑+0.32	106	5.6	83.25 ↑+0.06	106	5.2	79.80 ↑+0.76	106	5.9
(E–GRPO)	94.23 ↑+0.97	105	5.0	51.95 ↑+0.43	106	5.4	91.50 ↑+0.47	105	5.2	83.35 ↑+0.16	105	5.0	80.20 ↑+1.16	105	5.4
AgentAsk (Qwen–3–4B)															
SFT	93.99 ↑+0.73	105	5.3	51.90 ↑+0.38	106	5.2	91.45 ↑+0.42	105	5.1	83.10 ↑+0. -09	104	4.8	80.10 ↑+1.06	104	4.9
(E–GRPO)	94.72 ↑+1.46	103	4.2	52.07 ↑+0.55	104	4.6	91.55 ↑+0.52	104	4.5	82.86 ↓-0.33	103	4.1	80.64 ↑+1.60	103	4.3

Table 9: **MasRouter: ablations and efficiency across five benchmarks.**