

MAS-ZERO: DESIGNING MULTI-AGENT SYSTEMS WITH ZERO SUPERVISION

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) leveraging the impressive capabilities of Large Language Models (LLMs) hold significant potential for tackling complex tasks. However, most current MAS depend on manually designed agent roles and communication protocols. These manual designs often fail to align with the underlying LLMs' strengths and struggle to adapt to novel tasks. Recent automatic MAS approaches attempt to mitigate these limitations but typically necessitate a validation set for tuning and yield static MAS designs lacking adaptability during inference, while also removing the flexibility to reduce to simpler systems. We introduce MAS-ZERO, the first self-evolved, inference-time framework for automatic MAS design. MAS-ZERO employs meta-level design to iteratively design, critique, and refine MAS configurations tailored to *each problem instance*, without requiring a validation set. Critically, it enables dynamic problem decomposition and agent composition through meta-feedback on solvability and completeness, and reduction to simpler systems when appropriate. Experiments across reasoning (math and graduate-level QA), coding, and agentic (search-based) benchmarks, using both closed-source and open-source LLM backbones of varying sizes, demonstrate that MAS-ZERO outperforms strong manual and automatic MAS baselines. It achieves substantial average accuracy improvements of up to 16.69% on reasoning, 16.66% on coding, and 5.45% on agentic tasks, while maintaining cost efficiency.

1 INTRODUCTION

While standalone large language models (LLMs) have demonstrated strong performance across numerous tasks (DeepSeek-AI, 2025; Ke et al., 2025b; Vu et al., 2024), many problems remain too intricate for a single model to solve effectively (Wang et al., 2024b; Guo et al., 2024). To tackle these challenges, the exploration of multi-agent systems (MAS) composed of multiple LLM agents has gained increasing traction among researchers (Ke et al., 2025a).¹ These agents often assume distinct *roles*, such as generator or verifier (Shinn et al., 2024), engage in debates offering varied perspectives (Qian et al., 2025; Wang et al., 2024a), and perform assigned sub-tasks (Li et al., 2025).

A fundamental challenge in MAS lies in designing an effective connection and configuration of these agents to solve a given problem. Initially, MAS were handcrafted, with humans designing both agent roles and inter-agent communication protocols. However, MAS composed entirely of such manually designed configurations have faced issues such as poor problem specification and inter-agent misalignment (Cemri et al., 2025; Qiao et al., 2025), especially when the MAS agents are not specifically trained with such configurations.

These shortcomings are understandable, as manually specifying a MAS is difficult when the human designer and the underlying LLMs are not well aligned. Moreover, manual approaches do not scale well to novel problems, especially as the problems become more complex. Recent work has explored automatic MAS design, but they have significant limitations: (1) Most rely on a “training” phase with labeled validation sets to tune configurations, which are often unavailable in real-world scenarios and may not generalize. This training, based solely on *outcome* correctness, provides limited insight into

¹ Agents in a MAS can interact with external environmental tools e.g., search tools (Jiang et al., 2024), or collaborate with other agents to address tasks requiring diverse capabilities or multiple steps (Liang et al., 2023; Chen et al., 2024b). This work focuses on the latter scenario, where each agent within the MAS is an LLM communicating with other LLM agents.

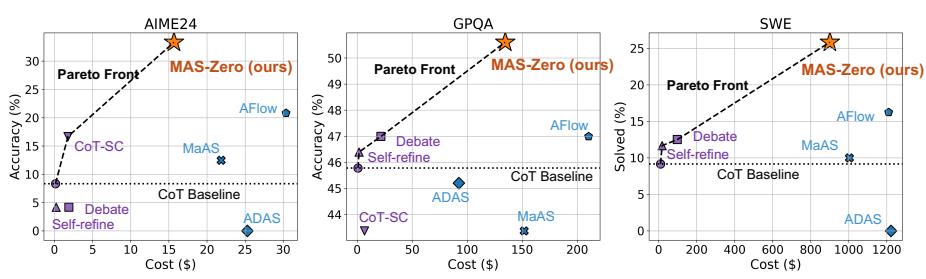


Figure 1: Scatter plots comparing the Pareto fronts of various GPT-4o-based systems on three benchmarks, including manual MAS, automatic MAS and MAS-ZERO. MAS-ZERO delivers high performance at lower cost than comparable automatic MAS methods, establishing a new frontier for accuracy vs. cost trade-off.

the system’s internal dynamics. (2) This reliance on validation sets often yields a *fixed* architecture (i.e., one for the entire problem set) which lacks per-problem adaptability at test time. (3) Even worse, these methods eliminate critical dynamics: they cannot reduce to a simpler MAS or a single-agent system when such strategies would be stronger (Huang et al., 2024), nor can they flexibly decompose a problem into smaller, more manageable sub-tasks. This limitation is less apparent on simple tasks such as GSM8K and HumanEval, which are commonly used for MAS evaluation (Hu et al., 2025b; Zhang et al., 2025a;c) and where most methods already perform well. On more challenging tasks (e.g., AIME24), however, the inability to revert or decompose becomes critical: as shown in Fig. 1, many baselines show little to no improvement over simple CoT, meaning the integrated system does not even outperform a single component.

To overcome these limitations, we argue that an effective automatic MAS should satisfy three core desiderata: (1) be dynamic enough to both decompose complex problems into smaller sub-tasks and reduce to a single agent or a simple MAS when a sophisticated MAS is *not needed*; (2) learn the capabilities of the underlying LLMs, and automatically design MAS structures that are aligned with those capabilities; and (3) support adaptivity at inference time, so that MAS designs can be tailored *per problem instance* without relying on training or validation sets. To our knowledge, no existing automatic MAS framework satisfies all three desiderata simultaneously. In this work, we propose a novel automatic inference-time MAS optimization framework, called **MAS-ZERO**, which designs MAS with *zero* supervision, while satisfying all the aforementioned desiderata. In particular, MAS-ZERO introduces a **meta-agent** that iteratively learns the capabilities of individual agents and their combinations, and refines the MAS design accordingly, thus operating at the MAS-level rather than the agent level (hence “meta”). The meta-agent also verifies candidate answers drawn from both refined MAS designs and simpler MAS or single-agent systems, ensuring the dynamic reduction capability. This process operates entirely at test time, allowing for unique MAS designs *per-problem*.

To achieve this, MAS-ZERO tasks the meta-agent to iteratively *design* and *critique* the MAS, maintain an *experience library*, *refine* the design based on accumulated experience, and ultimately *verify* the candidate answers. Fig. 2 illustrates a conceptual overview and contrasts MAS-ZERO with both automatic and manual MAS designs. Specifically, MAS-ZERO involves three key steps:

- **Initializing building blocks (MAS-Init):** MAS-ZERO starts with established single-agent (e.g., CoT, Self-Consistency) and simple human-designed MAS strategies (e.g., Debate, Self-Refine), executing each to generate initial outputs that seed later steps.
- **Self-evolving with iterative refinement (MAS-Evolve):** The meta-agent iteratively *designs* and *critiques* MAS configurations, guided by feedback on *solvability* and *completeness*, while accumulating prior designs and feedback in an experience library for continual refinement.
- **Selecting the best candidate with self-verification (MAS-Verify):** From the pool of outputs, including both building blocks and refined MAS iterations, the meta-agent verifies and selects the most reliable solution, dynamically choosing between complex MAS and simpler strategies.

Evaluations across three challenging domains—reasoning (math and graduate-level QA), coding, and agentic (search-based), using both closed-source and open-source LLM backbones of varying sizes (including GPT-4o, 32B, and 70B models) demonstrate that MAS-ZERO consistently outperforms strong manual and automatic MAS baselines. It achieves substantial average accuracy improvements of up to 16.69% on reasoning, 16.66% on coding, and 5.45% on agentic tasks. It also consistently lies on the Pareto frontier of accuracy and cost (Fig. 1). While the inference-time mechanism incurs

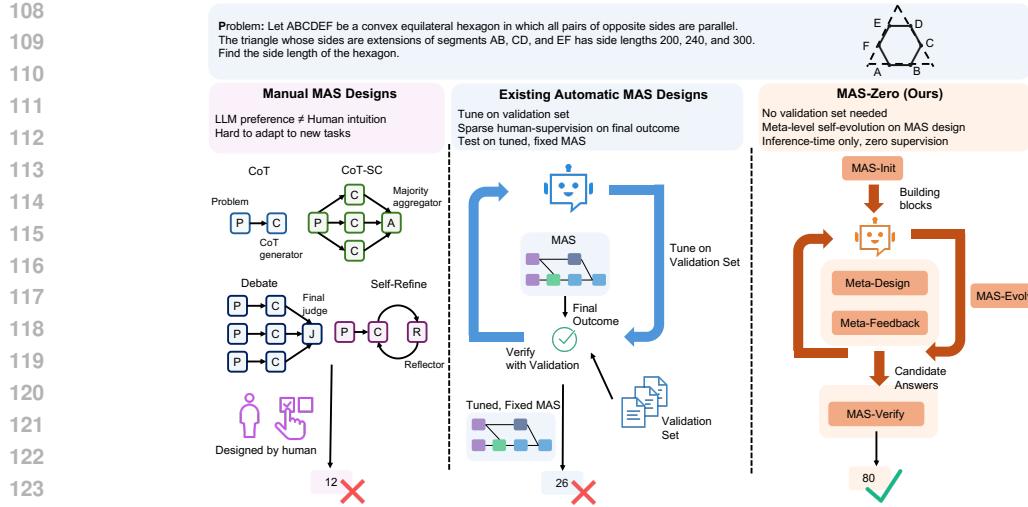


Figure 2: Conceptual comparison of MAS-ZERO, with existing automatic and manual MAS designs. MAS-ZERO avoids tuning MAS on validation set by maintaining a self-evolving process that iteratively designs and evaluates task-specific MAS at inference time.

higher token usage during testing, it avoids expensive validation-time optimization and shifts the design effort to the testing phase, where it can flexibly handle new tasks, and often be more effective (Agrawal et al., 2025). Such a trade-off has demonstrated significantly improved answers in this work and strong potential in the literature (Liu et al., 2024a). We believe that MAS-ZERO provides a complementary alternative for the MAS community, especially in scenarios where adaptability and generality outweigh the need for minimal token usage. In summary, our key contributions are:

- We introduce MAS-ZERO, to our knowledge, the **first inference-time-only** automatic MAS design framework. It works in a fully self-evolved way by learning from the behavior of the underlying LLM agents *at inference-time*, enabling per-instance adaptivity with *zero* supervision.
- We present a **new SoTA automatic MAS system** that achieves substantial performance gains over both manually designed and strong automatic baselines, while remaining cost-efficient and Pareto-optimal across diverse LLMs and domains.²
- We conduct a comprehensive evaluation of MAS-ZERO across diverse domains and LLMs, presenting **key insights**. For example, single-agent or simple MAS configurations can be surprisingly strong, in some cases even outperforming more sophisticated MAS designs. Crucially, MAS-ZERO is the only system that can *dynamically revert* to these simpler yet effective strategies, ensuring that such strengths are not overlooked.

2 RELATED WORK

Manual MAS design. Building on the success of single-agent systems (e.g., CoT (Wei et al., 2022), self-consistency (CoT-SC) (Wang et al., 2023a)), studies have shown that grouping multiple LLM agents into a MAS can substantially improve individual agent performance. To this end, a variety of manual-designed MAS approaches have been proposed (Xu et al., 2025; Zheng et al., 2024; Lu et al., 2025), including LLM debate (Du et al., 2023), and self-refine (Madaan et al., 2024). However, as discussed previously, these manual designs often suffer from limited adaptability and scalability, and their rigid structures may fail to align with the underlying strengths of LLMs.

Automatic MAS design. Recent work on automatic MAS design typically require validation set. We broadly categorize them into two families: **(1) val-pruning** starts with a fully connected, pre-defined graph of LLM agents or human-designed blocks and prune it based on validation performance. For example, MASS (Zhou et al., 2025a) uses rejection sampling, and MaAS (Zhang et al., 2025a) extends MASS with a question-wise masking mechanism to adapt subnetworks. However, their adaptability remains limited as the core MAS structure is constrained by the pre-defined structure, which is suboptimal for many tasks; **(2) val-generation** leverages a meta-agent LLM to generate MAS from

²We will open-source the data, code, and leaderboard for all components upon acceptance.

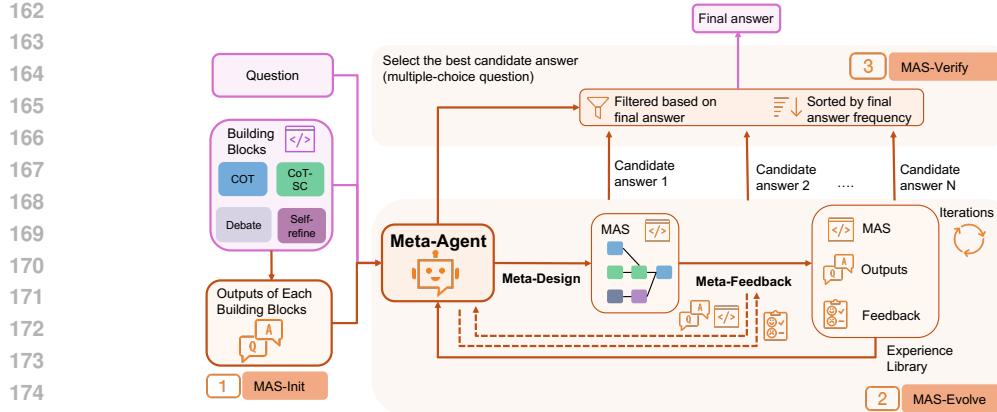


Figure 3: MAS-ZERO overview. Purple highlights the given input and final output. Orange highlights the components and steps in MAS-ZERO. Dashed arrows indicate the information flow *within* Meta-feedback. MAS-ZERO takes as inputs the question and building blocks, and solves the task in three key steps: **MAS-Init** (Sec. 3.1), **MAS-Evolve** (Sec. 3.2) and **MAS-Verify** (Sec. 3.3).

scratch, offering greater flexibility in defining novel agents and architectures compared to pre-defined structures. However, this expanded design space presents significant learning challenges. Recent efforts including ADAS (Hu et al., 2025b) and AFLow (Zhang et al., 2025c) frame MAS generation as a code generation task. ADAS stores and searches historical designs based on validation performance, while AFLow enhances this with Monte Carlo Tree Search. Our framework also represents MAS as executable code but differs fundamentally in several ways: instead of relying on potentially unreliable validation sets, MAS-ZERO uses a self-evolving process at inference time to learn the capabilities of agents for meta-level design. It further integrates question decomposition into MAS design, enabling MAS to be constructed and refined at the sub-task level. Finally, MAS-ZERO can dynamically revert to simpler building blocks when they are sufficient. These capabilities are not supported by existing automatic MAS systems. More methods like DyLAN (Liu et al., 2024b) are discussed in App. D.

3 MAS-ZERO FRAMEWORK

As shown in Fig. 3, MAS-ZERO first conducts **MAS-Init** (Sec. 3.1), where it executes each of the given building blocks. It then takes a question, the building blocks, and the outputs of each building block as inputs, ultimately producing the final answer. These inputs are processed by the central **meta-agent**, which orchestrates both the **MAS-Evolve** (Sec. 3.2) and **MAS-Verify** (Sec. 3.3) steps. Importantly, the whole process functions *without* prior knowledge or internal details of the underlying LLM agents. All steps are implemented through prompting and require only black-box access to LLM generation, making MAS-ZERO broadly applicable to any LLM without requiring fine-tuning or internal modifications. The corresponding pseudocode is provided in App. A.

3.1 MAS-INIT

MAS-Init serves as the entry point of MAS-ZERO by executing a set of predefined *building blocks*. These blocks correspond to established human-designed strategies (CoT, CoT-SC, Debate, and Self-Refine in this work) implemented as executable code. Given a question, MAS-Init runs each block to generate initial candidate solutions. These blocks and their outputs are used as: (1) input to the meta-agent for grounding the MAS design (Sec. 3.2), and (2) candidate answers that can be selected by the MAS-Verify (Sec. 3.3), enabling dynamic reduction to simpler MAS or single-agent systems.

3.2 MAS-EVOLVE

Given the question, the building blocks, and their outputs from MAS-Init, the meta-agent begins to design the MAS. Initially, it has no knowledge of the underlying LLM agents’ internal capabilities and may produce suboptimal designs. We propose an iterative process in which the meta-agent gradually learns the strengths of the component agents and refines its designs. This process alternates between two phases: (1) **meta-design** (Sec. 3.2.1), where the meta-agent decomposes the question into sub-tasks and proposes a MAS based on the building blocks and any accumulated *experience* from prior

216 iterations; **(2) meta-feedback** (Sec. 3.2.2), where the meta-agent reviews the proposed MAS and
 217 sub-tasks using intermediate outputs to assess their solvability and completeness, and then generates
 218 targeted feedback. The MAS, its intermediate outputs and feedback, are stored in an *experience*
 219 *library* that informs subsequent iterations. Through this cycle, the meta-agent progressively adjusts
 220 decompositions and configurations, yielding continual improvement without external supervision.
 221

222 3.2.1 META-DESIGN

223 The goal of this phase is to design a candidate MAS for the given task, which will then be reviewed in
 224 the next phase. Unlike existing work that tackles complex problems all at once, MAS-ZERO explicitly
 225 decomposes the original question into *manageable* yet *interdependent* sub-tasks. This decomposition
 226 not only breaks down complex problems into smaller parts but also creates opportunities to assign
 227 sub-task level MAS (i.e., *sub-MAS*) tailored to different components of the problem. For each sub-
 228 task, the meta-agent assigns a sub-MAS by modifying connections between given building blocks or
 229 adjusting their parameters (e.g., temperature, number of debate rounds, etc.). This deliberate design,
 230 informed by our preliminary experiments, balances exploration with improvement: the meta-agent is
 231 free to analyze questions, sub-tasks and assigned sub-MAS, but it should not arbitrarily invent new
 232 agents or blocks, nor prune the architecture without grounding in the provided building blocks.³
 233

234 3.2.2 META-FEEDBACK

235 **MAS and intermediate outputs.** Given the design produced in the meta-design phase, the meta-
 236 feedback phase reviews the MAS and generates feedback. Since a MAS is executable code, it can
 237 be run to obtain outputs, but relying only on the final answer is often sparse and uninformative.
 238 MAS-ZERO instead exploits the intermediate outputs, incorporating both *sub-task level* outputs from
 239 sub-MAS and *agent-level* outputs from individual LLMs. By jointly analyzing the final answer and
 240 these fine-grained signals, the meta-agent gains a much richer view of strengths and weaknesses
 241 across the MAS. Concretely, with the code-based representation, each sub-MAS is executed to solve
 242 its sub-task, producing intermediate outputs at two levels: the *sub-task (sub-MAS)* level and the *agent*
 243 level. These outputs form the basis for evaluation against the key criteria introduced below.
 244

245 **Criteria.** Given the above sub-task and agent level outputs, MAS-ZERO evaluates *solvability* and
 246 *completeness*. The meta-agent is given agency in determining each metric:

- 247 • **Solvability** requires that each sub-task be *independently* and *completely* solvable by its sub-MAS,
 248 ensuring that every sub-task yields reliable outputs.⁴
- 249 • **Completeness** requires that the complete set of sub-tasks covers all necessary information from
 250 the original input, ensuring that their answers can produce a correct and comprehensive aggregated
 251 answer to the original task. While an individual sub-task may address only part of the necessary
 252 content, all critical information must be processed and used at some point in the MAS.

253 **Generating feedback.** Based on the solvability and completeness, the meta-agent generates targeted
 254 natural language feedback on specific aspects of the MAS that may require revision. For example, if
 255 a sub-task is identified as not solvable, the feedback should suggest either further decomposing it or
 256 updating the corresponding sub-MAS in the next iteration. Conversely, if a sub-task is considered
 257 solvable, the feedback should indicate that it and its sub-MAS remain unchanged. Similarly, if the
 258 union of sub-tasks is found to miss necessary information, the feedback should recommend refining
 259 the decomposition of the original problem to incorporate the missing elements. Overall, this feedback
 260 guides subsequent meta-design iterations, allowing the overall system to iteratively converge toward
 261 an effective decomposition and MAS.

262 3.2.3 STORING THE EXPERIENCE AND REFINING THE DESIGN

263 After the first meta-design (Sec. 3.2.1), meta-feedback is collected (Sec. 3.2.2). The MAS, its interme-
 264 diate outputs, and the associated feedback are stored as *experience* in an *experience library*. In each
 265 subsequent iteration, meta-design is performed again, now with experience from the library provided
 266

267 ³To support code generation, we provide a template with utility functions and apply sanity checks (syntax
 268 validation and field consistency; see App. I).

269 ⁴To further aid the meta-agent, we allow each agent to output a special token, [T00 HARD], if it determines
 270 that the assigned sub-task is beyond its current capabilities.

270 as additional context to drive self-evolution. Through this process, the meta-agent dynamically adapts
 271 its decomposition strategy and sub-MAS assignments across iterations. This iterative accumulation
 272 of experience gives MAS-ZERO a persistent memory, enabling it to leverage knowledge from past
 273 iterations and build a stronger foundation for continual improvement.

274 As in many other self-evolving frameworks (Gao et al., 2025b), the meta-design and meta-feedback
 275 signals may be imperfect and ultimately depend on the underlying LLM. Nevertheless, we find em-
 276 pirically that MAS-ZERO allows initially imperfect designs to be progressively improved (Fig. G.1),
 277 and that our curated instruction design produces strong feedback—outperforming a simple ensemble
 278 alternative (Sec. 4.2). We view these results as a promising starting point and hope they inspire
 279 further research in advancing strategies for iterative MAS refinement

281 3.3 MAS-VERIFY

282 **Collecting candidate answers.** At each iteration of MAS-Evolve, the MAS is executed to produce
 283 intermediate outputs and a *candidate answer* (including both the chain-of-thought and the final
 284 answer). After multiple rounds, MAS-ZERO must determine which candidate answer is the most
 285 reliable and complete. Importantly, the pool of candidate answers includes *not only* those generated
 286 in each iteration of MAS-Evolve *but also* the outputs of the basic building blocks from MAS-Init.
 287 This design allows the meta-agent to select between them, leveraging the strong performance of
 288 simple strategies when they suffice, while also exploiting the complex MAS when needed.

289 **Verifying answers.** Relying on the last iteration (or any single iteration) is suboptimal due to
 290 stochastic LLM outputs and ongoing MAS refinement (ablations in Sec. 4.2). Instead, MAS-ZERO
 291 formulates verification as a *selection* problem and tasks the meta-agent with selecting the most
 292 coherent and correct output from the set of candidate answers, which is often more tractable than
 293 independently scoring each output (Gu et al., 2025; Zhou et al., 2025b), especially for challenging
 294 questions where correctness is hard to assess in isolation. Specifically, MAS-ZERO first *ranks*
 295 candidates by their final answer frequency. This acts as a prior favoring majority responses, a strategy
 296 shown to be effective in prior work (Wang et al., 2023a). It then *filters* out clearly invalid answers
 297 (e.g., not among the given options). Finally, it *selects* the best answer from the remaining candidates.

299 4 EXPERIMENTS

300 **Setup.** We consider both the closed-source **GPT-4o** (OpenAI, 2023) (web-search version for agentic
 301 tasks) and the open-source LLMs, **Llama3.3-70B-inst** (Llama, 2024) and **Qwen2.5-32B-inst** (Qwen,
 302 2025). To fairly evaluate how well MAS-ZERO performs relative to the underlying LLM used to
 303 construct the MAS, we always use the *same* LLM for both the meta-agent and individual agents
 304 (heterogeneous settings in Sec. 4.2). We use the same prompt template for all the tasks (App. H and I)
 305 and conduct 5 MAS-Evolve iterations (the maximum permitted by context-length). Together with the
 306 4 building blocks in MAS-Init, this yields 9 candidate answers, from which the meta-agent selects
 307 one final answer with MAS-Verify.

308 **Benchmarks.** We consider 2 **reasoning** benchmarks across math and science: **AIME24** (AIME,
 309 2024) and GPQA-diamond (**GPQA**) (Rein et al., 2023) (graduate-level QA), 1 **coding** benchmark
 310 **SWE-Bench-Lite-Oracle** (**SWE**) (Jimenez et al., 2024),⁵ and 2 search-based **agentic** benchmarks:
 311 **BrowseComp** (Wei et al., 2025) and **Frames** (Krishna et al., 2025).⁶ Existing automatic MAS
 312 methods largely restrict their evaluations to relatively simple reasoning tasks. To our knowledge,
 313 MAS-ZERO is the first to conduct evaluations on challenging reasoning, coding and agentic tasks.

314 **Baselines.** We include 2 widely used **single-agent** baselines: **CoT** (Wei et al., 2022) and self-
 315 consistency (**CoT-SC**) (Wang et al., 2023a); 6 **manual MAS** baselines: **Debate** (Du et al., 2023),
 316 **Self-refine** (Madaan et al., 2024), **ReConcile** (Chen et al., 2024b), **MAD** (Liang et al., 2023), **SPP**
 317 (Wang et al., 2023b) and **DyLAN** (Liu et al., 2024b). Note that CoT, CoT-SC, Debate and Self-Refine
 318 also serve as the building blocks in MAS-Init, allowing us to clearly observe how our system improves
 319 upon the initial configurations. For **automatic MAS**, we include 3 strong methods: **val-pruning**
 320 **MaAS** (Zhang et al., 2025a) and **val-generation ADAS** (Hu et al., 2025b) and **AFlow** (Zhang et al.,
 321 2025c). We also include the latest **training-based** method **MAS-GPT** (Ye et al., 2025).

322 ⁵Note that MAS-Verify does not apply to SWE, as correctness in SWE is determined directly by the compiler.

323 ⁶Benchmark statistics and more implementation details can be found in App. B.

LLMs Methods	GPT-4o			Llama3.3-70B			Qwen2.5-32B		
	AIME24	GPQA	Avg.	AIME24	GPQA	Avg.	AIME24	GPQA	Avg.
CoT	8.33	45.78	27.06 ^{↑14.91}	16.67	50.60	33.63 ^{↑11.32}	12.50	50.00	31.25 ^{↑9.24}
CoT-SC	16.67	43.37	30.02 ^{↑11.95}	29.17	51.20	40.18 ^{↑4.77}	16.67	49.40	33.04 ^{↑7.46}
Debate	4.17	46.99	25.58 ^{↑16.39}	20.83	50.60	35.72 ^{↑9.24}	8.33	49.40	28.87 ^{↑11.63}
Self-Refine	4.17	46.39	25.28 ^{↑16.69}	29.17	54.22	41.69 ^{↑3.26}	16.67	50.60	33.64 ^{↑6.86}
ReConcile	12.50	48.43	30.47 ^{↑11.50}	33.33	47.17	40.25 ^{↑4.71}	12.50	47.17	29.84 ^{↑10.66}
MAD	13.89	52.01	32.95 ^{↑9.02}	29.17	52.61	40.89 ^{↑4.07}	16.67	43.57	30.12 ^{↑10.37}
SPP	9.72	49.80	29.76 ^{↑12.21}	26.39	46.18	36.29 ^{↑8.67}	19.44	42.77	31.11 ^{↑9.39}
DyLAN	11.11	46.99	29.05 ^{↑12.92}	29.17	41.57	35.37 ^{↑9.59}	20.83	42.57	31.70 ^{↑8.79}
MaAS	12.50	43.37	27.94 ^{↑14.03}	33.33	43.98	38.65 ^{↑6.30}	20.83	46.99	33.91 ^{↑6.58}
ADAS	×	45.20	×	8.30	53.60	30.95 ^{↑14.01}	12.50	47.00	29.75 ^{↑10.74}
AFlow	20.83	46.99	33.91 ^{↑8.05}	33.33	47.59	40.46 ^{↑4.49}	33.33	46.39	39.86 ^{↑0.63}
MAS-GPT	13.89	43.98	28.94 ^{↑13.03}	26.39	40.00	33.20 ^{↑11.76}	23.61	37.35	30.48 ^{↑10.01}
MAS-ZERO	33.33	50.60	41.97	37.50	52.41	44.96	29.17	51.81	40.49

Table 1: Reasoning tasks results. “×” indicates 0% accuracy for MAS selected using the validation set. “↑” denotes the difference (improvement) that MAS-ZERO achieves compared to the baselines. Highlighting indicates single-agent, manual MAS, val-pruning automatic MAS, val-generation automatic MAS, training-based automatic MAS and our method. To fairly compare with validation-based baselines, we split each benchmark’s original test set into 20% for validation and 80% for testing. For methods do not use validation sets (including MAS-ZERO), we evaluate on the same 80% split. Standard deviations are given in App. E.

4.1 OVERALL RESULTS

Tables 1-3 show the results for reasoning, coding and agentic tasks across applicable LLMs and benchmarks. On average, MAS-ZERO achieves the *best* performance across all LLMs and domains. Below, we summarize the additional takeaways from the comparison:

Reasoning Tasks. From Table 1, we observe that (1) MAS-ZERO *consistently outperforms* all *automatic MAS* methods. Across all LLM backbones and benchmarks, it surpasses SoTA baselines, exceeding the strongest baseline, AFlow, by 13.03% on average with GPT-4o as the backbone. The only instance where it falls behind is on AIME24 with the Qwen backbone, where it underperforms AFlow by merely one sample (out of 24 total). Notably, ADAS fails completely on AIME24 (0% accuracy), despite having access to a validation set, underscoring the unreliability of validation-based baselines. (2) MAS-ZERO also *consistently outperforms* strong single-agent and manual MAS, with only two exceptions: GPQA with MAD using GPT-4o, and GPQA with Self-Refine using Llama.

Alarmingly, several automatic MAS baselines underperform manual MAS across multiple benchmarks. For example, CoT and CoT-SC consistently outperform MAS-GPT, ADAS, and MaAS. This further highlights the necessity of MAS-Init in MAS-ZERO, as it allows the system to dynamically revert to simpler strategies when a sophisticated MAS is not needed.

Coding Tasks. Similar to reasoning tasks, in Table 2 we observe MAS-ZERO clearly outperforms single-agent, manual and automatic MAS. Notably, it comes with 58% (GPT-4o) and 149% (Llama) relative gains over the strongest baseline AFlow. These margins exceed those observed in reasoning tasks, highlighting the effectiveness of MAS-ZERO in challenging tasks.

Agentic Tasks. We use GPT-4o with search as individual agent, which can query the internet and conduct multi-turn autonomous reasoning internally (meta-agent is still GPT-4o). From Table 3, we see that on average, MAS-ZERO continues to improve upon the basic building blocks. On Frames, MAS-ZERO underperforms Debate and Self-Refine. We speculate that when the search agent makes mistakes, those errors are grounded in retrieved content, making them more difficult to detect during MAS-Verify, leading to incorrect judgments. This highlights the importance of further strengthening the verifier in MAS-Verify (see Sec. 4.2 for more analysis).

LLMs Methods	GPT-4o		Llama3.3	
	SWE	SWE	SWE	SWE
CoT	9.17 ^{↑16.66}	2.92 ^{↑13.82}		
Debate	12.50 ^{↑13.33}	6.67 ^{↑10.07}		
Self-Refine	11.67 ^{↑14.16}	1.67 ^{↑15.07}		
MaAS	10.00 ^{↑15.83}	5.00 ^{↑11.74}		
AFlow	16.25 ^{↑9.58}	6.67 ^{↑10.07}		
MAS-ZERO	25.83	16.74		

Table 2: SWE results. Methods that cannot be adapted to SWE are not included. Qwen is not included due to its small maximum context length (32K).

LLM Methods	GPT-4o w/ search		
	BrowseComp	Frames	Avg.
CoT	3.97	59.76	31.86 ^{↑5.45}
CoT-SC	8.66	63.58	36.12 ^{↑1.19}
Debate	3.94	70.45	37.19 ^{↑0.12}
Self-Refine	5.51	67.89	36.70 ^{↑0.61}
MAS-ZERO	9.45	65.18	37.31

Table 3: Results on agentic tasks.

378 **Cost-efficiency.** Fig. 1 shows the trade-off between performance and cost for GPT-4o across three
 379 benchmarks. Cost is computed using the official OpenAI API pricing⁷ and includes both “training”
 380 (if any) and test-time usage. We observe that MAS-ZERO **lies on the Pareto front across all**
 381 **three datasets.** It is significantly more cost-efficient than AFlow, MaAS, and ADAS, with the lone
 382 exception of ADAS on GPQA, where the cost increase comes with a 12% accuracy improvement.
 383 Of automatic MAS frameworks, MAS-ZERO delivers the highest performance at relatively low
 384 cost. While it is expected that automatic MAS methods incur higher costs than manual baselines,
 385 MAS-ZERO delivers substantially better performance, making the trade-off highly favorable.

386 4.2 FURTHER ANALYSIS AND ABLATIONS

387 While Sec. 4.1 establishes the overall effectiveness of MAS-ZERO across domains and LLMs, here we
 388 analyze the role of the **meta-agent** and **each of the three steps** through a series of targeted ablations.
 389 The results, detailed below, show that a capable meta-agent consistently enhances performance and
 390 that all three steps contribute meaningfully and complementarily to the final improvements.

391 **Diverse meta-agents.** While MAS-ZERO shows strong
 392 performance across various LLMs, we further examine
 393 whether weaker or stronger LLMs can effectively serve as
 394 meta-agents. For stronger LLM, we conduct experiments
 395 with a reasoning LLM, **o3-mini** (OpenAI, 2025b). As
 396 shown in Table 4, MAS-ZERO outperforms the consid-
 397 ered baselines on average, indicating that the benefits of
 398 MAS-ZERO generalize well across model strengths. For
 399 weaker LLMs, we conduct experiments with **GPT-OSS-
 400 20B** (OpenAI et al., 2025), **Qwen2.5-7B** (Qwen, 2025), and **Qwen2.5-Coder-3B** (Hui et al., 2024),
 401 **GPT-4.1-nano** (OpenAI, 2025a). We find that these models are unable to reliably follow instructions
 402 and often produce syntactically incorrect Python code, suggesting that the meta-agent role requires
 403 sufficiently strong capabilities to handle its multiple responsibilities.

404 **Heterogeneous agents.** The previous experiments use
 405 the same LLM for both the meta-agent and the individual
 406 agents and already achieved strong results. An intriguing
 407 question is whether heterogeneous assignments can yield
 408 additional benefits or drawbacks. Specifically, we explore
 409 pairing a stronger LLM as the meta-agent with a weaker
 410 LLM as the individual agent, and vice versa. As shown
 411 in Table 5, when GPT-4o is the individual agent and the meta-agent is replaced with o3-mini,
 412 performance improves notably but still falls short of directly using o3-mini for both roles. Conversely,
 413 when o3-mini is the individual agent and the meta-agent is replaced with GPT-4o, performance
 414 decreases, though it remains better than the setting where GPT-4o is the agent and o3-mini is the
 415 meta-agent. These results suggest that while a stronger meta-agent can provide benefits, the overall
 416 performance is ultimately constrained by the capability of the individual agent.

417 **MAS-Init.** Table 1 suggests that building blocks
 418 can achieve strong performance in some prob-
 419 lems. To quantify their contribution, we ablate
 420 MAS-Init by skipping execution of the building
 421 blocks in the first step, letting MAS-Verify judge
 422 *solely* based on the five candidate solutions pro-
 423 duced by the five iterations of MAS-Evolve. As
 424 shown in Table 6(-**MAS-Init**), this significantly
 425 degrades performance, highlighting the impor-
 426 tance of including MAS-Init and the ability of
 427 MAS-ZERO to dynamically revert to building blocks.

428 **MAS-Evolve.** To evaluate its importance, we first conduct an ablation by skipping the entire MAS-
 429 Evolve and letting MAS-Verify judge *solely* based on the four building block outputs from MAS-Init.
 430 As shown in Table 6(-**MAS-Evolve**), the performance drops notably, indicating that MAS-Evolve

LLM Methods	o3-mini		
	AIME24	GPQA	Avg.
CoT	70.00	72.22	71.11 ^{↑12.27}
CoT-SC	80.00	72.73	76.36 ^{↑7.02}
Debate	86.67	77.78	82.22 ^{↑1.16}
Self-Refine	76.67	74.24	75.45 ^{↑7.93}
MAS-ZERO	90.00	76.77	83.38

Table 4: MAS-ZERO with stronger agents.

Agent	Meta-agent	AIME24	GPQA	Avg.
GPT-4o	GPT-4o	33.33	50.60	41.97
o3-mini	o3-mini	90.00	76.77	83.38
GPT-4o	o3-mini	36.67	60.10	48.38
o3-mini	GPT-4o	83.33	73.74	78.54

Table 5: MAS-ZERO with different models.

LLM Methods	GPT-4o		
	AIME24	GPQA	Avg.
MAS-ZERO	33.33	50.60	41.97
- MAS-Init	12.50	48.43	30.46 ^{↓11.50}
- MAS-Evolve	20.00	48.73	34.37 ^{↓7.60}
- meta-design	20.83	45.18	33.01 ^{↓8.96}
- meta-feedback	25.00	42.17	33.59 ^{↓8.38}
→ ensemble meta-feedback	16.67	46.88	31.77 ^{↓10.19}
- MAS-Verify	6.70	33.83	20.27 ^{↓21.70}

Table 6: Ablations on the three steps in MAS-ZERO.

⁷More details are given in App. C

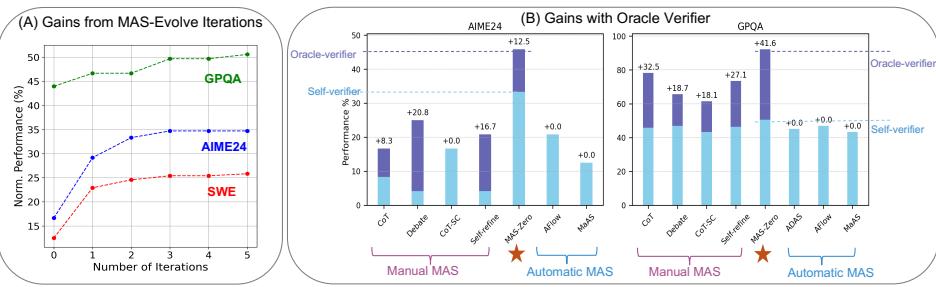


Figure 4: (A) Performance gains (GPT-4o) over MAS-Evolve. (B) Performance gains (GPT-4o) given an oracle verifier. Automatic MAS baselines cannot integrate external verifiers, yielding zero improvement.

is useful for the overall improvements. For **meta-design**, we modify the prompt to ask the meta-agent to propose a MAS configuration *without* attempting to decompose the question into sub-tasks; Table 6(-**meta-design**) shows that removing decomposition leads to a significant performance drop, demonstrating that breaking down the task is a meaningful contributor to the effectiveness of MAS-ZERO. For **meta-feedback**, we test two variants: (1) modifying the prompt so that the meta-agent critiques the current MAS *without* analyzing the solvability and completeness of each sub-task or LLM agent (-**meta-feedback**); (2) since meta-feedback can be noisy due to the self-evolving nature of the system, we explore a straightforward method to improve reliability via ensembling (→ **ensemble meta-feedback**). Following Du et al. (2023), we generate multiple feedback candidates (three in our experiments) from the meta-agent and then use an additional call to the meta-agent to select the best one. The corresponding rows in Table 6 reveal that removing meta-feedback results in a large performance drop, confirming that it is critical to the overall effectiveness of MAS-ZERO. Surprisingly, the ensemble approach not only fails to improve performance but even reduces it. This counter-intuitive result suggests that the current straightforward meta-feedback is already strong, and advancing MAS-ZERO will require designing more principled strategies for reliable feedback.

Gains from MAS-Evolve at each iteration. To further evaluate the self-evolving capability in MAS-Evolve, we examine performance across iterations. As shown in Fig. 4(A), accuracy at iteration 0 (before MAS-Evolve, only MAS-Init) and 1 (after the first refinement) is notably lower, indicating that the system struggles to design effective MAS at the outset. With subsequent iterations, however, MAS-ZERO progressively improves, demonstrating a strong ability to self-evolve through the refinement cycle of meta-design, meta-feedback, and the accumulated experience library.

MAS-Verify. This final step determines which candidate solution is selected as the final answer. To assess its importance, we conducted an ablation where the system simply used the *last* iteration as the output, without any additional judgment. The last row in Table 6 shows a sharp performance decline (the largest drop among all ablations). This is intuitive because, as shown earlier, the ability to dynamically revert to building blocks is indispensable. Yet it is also revealing, since the self-evolving nature of MAS-ZERO might suggest that the final iteration should yield the strongest solution. Instead, the outcome highlights that iterative refinement alone is insufficient, and that effective verification is essential to counteract the stochasticity of pure self-evolving methods without ground-truth signals.

Potential of MAS-Verify with oracle verifier. We showed that MAS-Verify is crucial, and this highlights substantial headroom for further improvement. Unlike existing automatic MAS frameworks, MAS-ZERO can seamlessly incorporate *external verifiers*, making it naturally positioned to benefit from advances in verification techniques. Fig. 4(B) illustrates this potential: when equipped with an *oracle verifier* that labels outputs as “correct” or “incorrect” using ground-truth answers, MAS-ZERO’s performance improves dramatically, further widening the gap over both manual and automatic MAS and pushing GPQA close to 95%. This demonstrates that stronger verification could unlock significant headroom for MAS-ZERO.

5 CONCLUSION

We presented MAS-ZERO, the first *inference-time-only* automatic MAS design framework with *zero* supervision. It iteratively designs and refines MAS, decomposes complex questions, reverts to simpler strategies when sufficient, and verifies candidate answers. Comprehensive experiments show its strong effectiveness, cost-efficiency, and the contribution of each step.

486 REFERENCES
487

488 Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong,
489 Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik
490 Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa:
491 Reflective prompt evolution can outperform reinforcement learning, 2025. URL <https://arxiv.org/abs/2507.19457>.
492

493 Team AIME. Aime problems and solutions, 2024. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.
494

495 Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari,
496 Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E.
497 Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL <https://arxiv.org/abs/2503.13657>.
498

499 Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and
500 Yemin Shi. Autoagents: A framework for automatic agent generation, 2024a. URL <https://arxiv.org/abs/2309.17288>.
501

502 503 Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-table conference improves
504 reasoning via consensus among diverse LLMs. In *Proceedings of the 62nd Annual Meeting of
505 the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024b. URL <https://arxiv.org/abs/2309.13007>.
506

507 508 Justin Chih-Yao Chen, Sukwon Yun, Elias Stengel-Eskin, Tianlong Chen, and Mohit Bansal. Sym-
509 bolic mixture-of-experts: Adaptive skill-based routing for heterogeneous reasoning, 2025. URL
510 <https://arxiv.org/abs/2503.05641>.
511

512 Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu,
513 Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong
514 Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent
515 behaviors, 2023. URL <https://arxiv.org/abs/2308.10848>.
516

517 Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang,
518 Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent
519 collaboration via evolving orchestration, 2025. URL <https://arxiv.org/abs/2505.19591>.
520

521 DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,
522 2025. URL <https://arxiv.org/abs/2501.12948>.
523

524 Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving
525 factuality and reasoning in language models through multiagent debate, 2023. URL <https://arxiv.org/abs/2305.14325>.
526

527 Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rock-
528 täschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first
529 International Conference on Machine Learning*, 2024. URL <https://openreview.net/forum?id=9ZxnPZGmPU>.
530

531 Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin,
532 and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents, 2025a. URL <https://arxiv.org/abs/2504.15257>.
533

534 Huanang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong
535 Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang,
536 Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian,
537 Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A
538 survey of self-evolving agents: On path to artificial super intelligence, 2025b. URL <https://arxiv.org/abs/2507.21046>.
539

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen,
Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni,
and Jian Guo. A survey on llm-as-a-judge, 2025. URL <https://arxiv.org/abs/2411.15594>.

540 Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest,
 541 and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and
 542 challenges. *arXiv preprint arXiv:2402.01680*, 2024.

543 Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan
 544 Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping
 545 Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in
 546 real-world task automation, 2025a. URL <https://arxiv.org/abs/2505.23885>.

547 Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth
 548 International Conference on Learning Representations*, 2025b. URL <https://openreview.net/forum?id=t9U3LW7JVX>.

549 Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and
 550 Siheng Chen. Self-evolving multi-agent collaboration networks for software development, 2024.
 551 URL <https://arxiv.org/abs/2410.16946>.

552 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,
 553 and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth
 554 International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=Ikmd3fKBPQ>.

555 Binyuan Hui et al. Qwen2.5-coder: A technical report on code-specialized models. *arXiv preprint
 556 arXiv:2409.12186*, 2024. URL <https://arxiv.org/abs/2409.12186>.

557 Jinhao Jiang, Jiayi Chen, Junyi Li, Ruiyang Ren, Shijie Wang, Wayne Xin Zhao, Yang Song, and
 558 Tao Zhang. Rag-star: Enhancing deliberative reasoning with retrieval augmented verification and
 559 refinement. *arXiv preprint arXiv:2412.12881*, 2024.

560 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R
 561 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth
 562 International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

563 Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li,
 564 Chengwei Qin, Peifeng Wang, Silvio Savarese, Caiming Xiong, and Shafiq Joty. A survey of
 565 frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems, 2025a. URL
 566 <https://arxiv.org/abs/2504.09037>.

567 Zixuan Ke, Yifei Ming, Xuan-Phi Nguyen, Caiming Xiong, and Shafiq Joty. Demystifying domain-
 568 adaptive post-training for financial llms. *arXiv preprint arXiv:2501.04961*, 2025b.

569 Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vard-
 570 hamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller,
 571 Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into
 572 self-improving pipelines, 2023. URL <https://arxiv.org/abs/2310.03714>.

573 Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam
 574 Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-
 575 augmented generation, 2025. URL <https://arxiv.org/abs/2409.12941>.

576 Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning
 577 in multi-agent systems, 2025. URL <https://arxiv.org/abs/2410.02189>.

578 Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi,
 579 and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent
 580 debate. *arXiv preprint arXiv:2305.19118*, 2023.

581 Tongxuan Liu, Xingyu Wang, Weizhe Huang, Wenjiang Xu, Yuting Zeng, Lei Jiang, Hailong Yang,
 582 and Jing Li. Groupdebate: Enhancing the efficiency of multi-agent debate using group discussion.
 583 *arXiv preprint arXiv:2409.14051*, 2024a.

584 Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network
 585 for task-oriented agent collaboration, 2024b. URL <https://arxiv.org/abs/2310.02170>.

594 Team Llama. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
 595

596 Cong Lu, Shengran Hu, and Jeff Clune. Intelligent go-explore: Standing on the shoulders of giant
 597 foundation models, 2025. URL <https://arxiv.org/abs/2405.15143>.

598 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri
 599 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement
 600 with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

601

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

603 OpenAI. Openai gpt-4.1 nano model, 2025a. URL <https://platform.openai.com/docs/models/gpt-4.1-nano>. Released April 14, 2025 via API; 1 million token context window; fastest cheapest
 604 version in GPT-4.1 family.

605

606 OpenAI. Openai o3-mini system card, 2025b. URL <https://cdn.openai.com/o3-mini-system-card-feb10.pdf>. January 31, 2025.

607

608 OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin
 609 Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler
 610 Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen,
 611 Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives,
 612 Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher,
 613 Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar,
 614 Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman,
 615 Dominik Kundel, Jason Kwon, Volodymyr Krytylov, Elaine Ya Le, Guillaume Leclerc, James Park
 616 Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily,
 617 Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath,
 618 Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles,
 619 Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano,
 620 Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry
 621 Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu
 622 Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max
 623 Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey,
 624 Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin
 625 Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney,
 626 Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting
 627 Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b gpt-oss-20b
 628 model card, 2025. URL <https://arxiv.org/abs/2508.10925>.

629

630 Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du,
 631 Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based
 632 multi-agent collaboration, 2025. URL <https://arxiv.org/abs/2406.07155>.

633

634 Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun
 635 Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation, 2025. URL
 636 <https://arxiv.org/abs/2410.07869>.

637

638 Team Qwen. Qwen2.5 technical report, 2025. URL <https://arxiv.org/abs/2412.15115>.

639

640 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani,
 641 Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof qa benchmark,
 642 2023. URL <https://arxiv.org/abs/2311.12022>.

643

644 Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic
 645 ILM agent search in modular design space. *arXiv preprint arXiv:2410.06153*, 2024.

646

647 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion:
 648 Language agents with verbal reinforcement learning. *Advances in Neural Information Processing
 649 Systems*, 36, 2024. URL <https://arxiv.org/abs/2303.11366>.

650

651 Tu Vu, Kalpesh Krishna, Salaheddin Alzubi, Chris Tar, Manaal Faruqui, and Yun-Hsuan Sung.
 652 Foundational autoraters: Taming large language models for better automatic evaluation. *arXiv
 653 preprint arXiv:2407.10817*, 2024.

648 Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun
 649 Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. Rema: Learning to meta-think for llms with
 650 multi-agent reinforcement learning, 2025. URL <https://arxiv.org/abs/2503.09501>.

651 Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances
 652 large language model capabilities, 2024a. URL <https://arxiv.org/abs/2406.04692>.

653 Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the bounds
 654 of llm reasoning: Are multi-agent discussions the key?, 2024b. URL <https://arxiv.org/abs/2402.18272>.

655 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha
 656 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language
 657 models. In *The Eleventh International Conference on Learning Representations*, 2023a. URL
 658 <https://openreview.net/forum?id=1PL1NIMMrw>.

659 Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing
 660 cognitive synergy in large language models: A task-solving agent through multi-persona self-
 661 collaboration. *arXiv preprint arXiv:2307.05300*, 2023b. URL <https://arxiv.org/abs/2307.05300>.

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

665 Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won
 666 Chung, Alex Tachard Passos, William Fedus, and Amelia Glaeser. Browsecmp: A simple yet
 667 challenging benchmark for browsing agents, 2025. URL <https://arxiv.org/abs/2504.12516>.

668 Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong
 669 Mao. Expertprompting: Instructing large language models to be distinguished experts, 2025. URL
 670 <https://arxiv.org/abs/2305.14688>.

671 Rui Ye, Shuo Tang, Rui Ge, Yixin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training
 672 llms to build llm-based multi-agent systems, 2025. URL <https://arxiv.org/abs/2503.03686>.

673 Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards
 674 automatic multi-agent generation via evolutionary algorithms. *arXiv preprint arXiv:2406.14228*,
 675 2024.

676 Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng,
 677 Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for
 678 llm-based multi-agent systems, 2024. URL <https://arxiv.org/abs/2410.02506>.

679 Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent
 680 architecture search via agentic supernet, 2025a. URL <https://arxiv.org/abs/2502.04180>.

681 Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang,
 682 Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies
 683 via graph neural networks, 2025b. URL <https://arxiv.org/abs/2410.11782>.

684 Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen
 685 Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin
 686 Wu. Aflow: Automating agentic workflow generation, 2025c. URL <https://openreview.net/forum?id=z5uVAKwmjf>.

687 Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le,
 688 and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models,
 689 2024. URL <https://arxiv.org/abs/2310.06117>.

690 Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and
 691 Sercan Ö. Arik. Multi-agent design: Optimizing agents with better prompts and topologies, 2025a.
 692 URL <https://arxiv.org/abs/2502.02533>.

702 Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Ed H. Chi, Denny
703 Zhou, Swaroop Mishra, and Steven Zheng. SELF-DISCOVER: Large language models self-
704 compose reasoning structures. In *The Thirty-eighth Annual Conference on Neural Information
705 Processing Systems*, 2024. URL <https://openreview.net/forum?id=BROvXhmzYK>.

706 Yilun Zhou, Austin Xu, Peifeng Wang, Caiming Xiong, and Shafiq Joty. Evaluating judges as
707 evaluators: The jets benchmark of llm-as-judges as test-time scaling evaluators. *arXiv preprint
708 arXiv:2504.15253*, 2025b.

709 Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbulin, and Jürgen
710 Schmidhuber. Gptswarm: Language agents as optimizable graphs, 2024. URL <https://arxiv.org/abs/2402.16823>.

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

A MAS-ZERO ALGORITHM

In Section 3, we detail the three steps of MAS-ZERO. Algorithm 1 presents the detailed algorithm. Highlighting indicates MAS-Init, MAS-Evolve and MAS-Verify.

Algorithm 1: MAS-ZERO: Designing Multi-Agent Systems with Zero Supervision

Input: Question Q , building blocks $\{\mathcal{M}^{(1)}, \dots, \mathcal{M}^{(k)}\}$, meta-agent \mathcal{A} , iterations T
Output: Final Answer y^*
Initialize candidate answers $\mathcal{H} \leftarrow []$, experience library $\mathcal{E} \leftarrow \emptyset$;

Step 1: MAS-Init

```

foreach building block  $\mathcal{M}^{(i)}$  do
   $Y_0^{(i)} \leftarrow \text{Execute}(\mathcal{M}^{(i)}, Q)$ ; // Run each building block
  extract final answer  $y_0^{(i)}$  from  $Y_0^{(i)}$ ;
  append  $y_0^{(i)}$  to  $\mathcal{H}$ ;
   $\mathcal{E} \leftarrow \mathcal{E} \cup \{(Q, \mathcal{M}^{(i)}, y_0^{(i)})\}$ ; // Store the answers from MAS-Init

```

Step 2: MAS-Evolve

```

 $(\mathcal{Q}_0, \mathcal{M}_0) \leftarrow \mathcal{A}.\text{Meta\_Design}(Q, \{\mathcal{M}^{(i)}\}, \mathcal{E}, \text{Constraints} = \{\mathcal{M}^{(i)}\})$ ;
// Decompose into sub-tasks  $\mathcal{Q}_0$  and assign sub-MAS  $\mathcal{M}_0$  grounded in building blocks
for  $t = 1$  to  $T$  do
   $Y_t \leftarrow \text{Execute}(\mathcal{M}_{t-1}, \mathcal{Q}_{t-1})$ ; // Run current MAS on sub-tasks
  extract sub-task outputs  $\{(x_j^{\text{sub}}, y_j^{\text{sub}})\}$  and agent outputs  $\{(x_\ell^{\text{agent}}, y_\ell^{\text{agent}})\}$  from  $Y_t$ ;
   $(\mathcal{Q}_t, \mathcal{M}_t, y_t, f_t) \leftarrow \mathcal{A}.\text{Meta\_Feedback}(Q, \mathcal{Q}_{t-1}, \mathcal{M}_{t-1}, \{(x_j^{\text{sub}}, y_j^{\text{sub}})\}, \{(x_\ell^{\text{agent}}, y_\ell^{\text{agent}})\}, \mathcal{E}, \text{Constraints} = \{\mathcal{M}^{(i)}\})$ ;
  // Assess solvability and completeness; revise decomposition and sub-MAS
   $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathcal{Q}_{t-1}, \mathcal{M}_{t-1}, Y_t, f_t)\}$ ; // Store sub-tasks, sub-MAS, intermediate outputs, and feedback
  if  $y_t \neq \perp$  then
     $\mathcal{H}$  append  $y_t$  to  $\mathcal{H}$ 

```

Step 3: MAS-Verify

```

 $y^* \leftarrow \mathcal{A}.\text{Self\_Verify}(\mathcal{H})$ ; // Select final answer from all candidates
return  $y^*$ ;

```

B IMPLEMENTATIONS, BENCHMARKS AND BASELINES DETAILS

Implementation details. As described in Sec. 4, MAS-ZERO produces 9 candidate answers. For fair comparison, we sample 9 independent outputs for CoT-SC and take the majority vote. Similarly, both debate and self-refine are run for 9 rounds. All models are accessed through their respective APIs.⁸ Temperature for meta-agent is set to 0.5. For baselines, we strictly use parameters found in original papers and provided code.

Benchmarks Table B.1 shows the detailed statistics for each dataset. For BrowseComp and Frames, we randomly sample 10% for testing, due to the large dataset size. We evaluate SWE using its official code available at <https://github.com/SWE-bench/SWE-bench/>.

Split	AIME24	GPQA	SWE	Browse	Comp	Frames
Validation	6	32	60	—	—	—
Test	24	166	240	126	—	82

Table B.1: Data size for each split in each dataset.

Baselines details We use the official implementations of all baselines, sourced directly from their public repositories. For manual MAS methods, this includes ReConcile (<https://github.com/dinobby/ReConcile>), MAD (<https://github.com/Skytliang/Multi-Agents-Debate>), SPP (<https://github.com/MikeWangWZHL/Solo-Performance-Prompting>), and DyLAN (<https://github.com/SALT-NLP/DyLAN>). For automatic MAS methods, this includes ADAS (<https://github.com/ShengranHu/ADAS>), AFlow (<https://github.com/FoundationAgents/MetaGPT/tree/main/examples/aflow>), and MaAS (<https://github.com/bingreeky/MaAS>).

⁸We use TogetherAI API (<https://www.together.ai/>) for Llama and Qwen.

810
811 C COST COMPUTATION

812 In Fig. 1, we report the cost of single-agent systems, manual MAS, automatic MAS, and MAS-
 813 ZERO. Costs are computed using OpenAI’s official pricing as of May 2025 at <https://openai.com/api/pricing/>. To ensure accuracy, we track usage directly via the official OpenAI API field
 814 “response.usage” for all methods. As a result, the reported values reflect the actual monetary cost,
 815 accounting for both input and output tokens.
 816

817
818 D ADDITIONAL RELATED WORK
819

820 In Sec. 2, we briefly introduced the most important related works and highlighted their contrast
 821 with MAS-ZERO. In this section, we provide a more detailed discussion of existing works. For
 822 completeness, we also note that a number of training-based approaches have been proposed, but we
 823 omit them from Sec. 2 since MAS-ZERO does not involve updating LLM parameters.
 824

825 Some prior work treats prompt optimization for individual agents as part of MAS design. Examples
 826 include PromptBreeder (Fernando et al., 2024), DsPy (Khattab et al., 2023) and Self-Discover (Zhou
 827 et al., 2024). More recently, this idea has been extended to broader automatic MAS design, where
 828 prompt optimization is included either as an additional design step or as part of the search space.
 829

830 **Manual MAS design.** In addition to the approaches discussed in Sec. 2, several other methods
 831 fall into this family. DyLAN Liu et al. (2024b) uses message passing to dynamically activate agent
 832 compositions; Reconcile (Chen et al., 2024b) and MAD (Liang et al., 2023) employ debate and
 833 round-table discussion, SPP (Wang et al., 2023b) leverages collaboration among multiple personas.
 834

835 **Automatic MAS design.** We follow the categories introduced in Sec. 2 and additionally include
 836 the training-based family.
 837

838 **Val-Pruning.** This line of work has evolved quickly (Zhang et al., 2024; 2025b; Hu et al., 2024).
 839 Earlier examples include GPTSwarm (Zhuge et al., 2024) which optimizes graph structures via
 840 reinforcement learning but struggles to represent workflows with conditional state due to limitations
 841 of static graphs. AgentSquare (Shang et al., 2024) leverages a verifier as a performance predictor to
 842 guide the pruning.
 843

844 **Val-Generation.** Besides the approaches introduced in Sec. 2 that employ a meta-agent to gen-
 845 erate building block connections and configurations, another line of work uses the meta-agent to
 846 directly generate the required agents or blocks. For example, AutoAgents (Chen et al., 2024a)
 847 and AgentVerse (Chen et al., 2023) create specialized agents through a planner agent, while EvoA-
 848 gent (Yuan et al., 2024) applies evolutionary algorithms to optimize this generation process. Similarly,
 849 Symbolic-MoE (Chen et al., 2025) leverages validation signals to construct block profiles and select
 850 the best-performing planner agent.
 851

852 **Training-based.** More recent work attempts to explicitly train the agents or meta-agents within MAS.
 853 For example, ReMA (Wan et al., 2025) and OWL (Hu et al., 2025a) train the agents in a manual MAS,
 854 while MAS-GPT (Ye et al., 2025) collects data from off-the-shelf MAS and trains a meta-agent via
 855 supervised fine-tuning (SFT). FlowReasoner (Gao et al., 2025a) builds on this by extending SFT with
 856 reinforcement learning (RL). Puppeteer (Dang et al., 2025) directly RL trains the meta-agent in an
 857 end-to-end manner.
 858

859
860 E STANDARD DEVIATION FOR THE EXPERIMENTS
861

862 To confirm the statistical significance of the experimental results in Table 1, we repeat the experiment
 863 *three* times, following (Zhang et al., 2025c; Liu et al., 2024b). We can see that the MAS can exhibit
 864 high variance due to the inherent nature of multi-agent systems: the variance may be amplified by the
 865 interactions among multiple agents (Cemri et al., 2025), and the generated temperature of the agents
 866 is typically non-zero. Nevertheless, MAS-ZERO is significantly stronger than other baselines, with
 867 only one exception in AIME24 and two in GPQA, as mentioned in Sec. 4.
 868

864	LLMs	GPT-4o	Llama3.3	Qwen2.5
865	CoT	± 1.97	± 1.96	± 0.00
866	CoT-SC	± 3.40	± 5.20	± 1.96
867	Debate	± 7.08	± 7.08	± 5.20
868	Self-Refine	± 3.93	± 1.97	± 1.97
869	ReConcile	± 1.97	± 1.96	± 1.97
870	MAD	± 1.96	± 3.40	± 3.40
871	SPP	± 5.20	± 1.96	± 5.89
872	DyLAN	± 1.96	± 0.00	± 3.93
873	ADAS	\times	± 5.30	± 6.38
874	AFlow	± 1.96	± 1.96	± 0.00
875	MAS-GPT	± 3.54	± 1.96	± 5.20
876	MAS-ZERO	± 5.89	± 3.15	± 5.20

Table E.1: Standard deviations for AIME24.

LLMs	GPT-4o	Llama3.3	Qwen2.5
CoT	± 1.29	± 0.89	± 2.72
CoT-SC	± 1.36	± 0.78	± 1.94
Debate	± 0.41	± 0.78	± 1.07
Self-Refine	± 2.08	± 2.08	± 3.01
ReConcile	± 1.29	± 2.43	± 1.85
MAD	± 1.28	± 1.28	± 3.08
SPP	± 1.74	± 1.25	± 1.20
DyLAN	± 3.44	± 0.49	± 0.75
ADAS	± 3.83	± 3.83	± 3.98
AFlow	± 1.70	± 2.48	± 1.77
MAS-GPT	± 1.02	± 1.02	± 2.25
MAS-ZERO	± 1.67	± 0.51	± 2.08

Table E.2: Standard deviations for GPQA.

F ILLUSTRATION OF MAS-EVOLVE

Fig. F.1 illustrates MAS-Evolve. Given a question and the building blocks, the meta-agent is prompted to decompose the task and propose a MAS (see Appendix H for detailed prompts). The meta-agent then generates a MAS in the form of code, which is executed by an external compiler to produce intermediate and final outputs for the sub-tasks and agents.

After this meta-design and execution, the meta-feedback phase begins. In this phase, both the resulting MAS and its intermediate outputs are provided to the meta-agent to review their *solvability* and *completeness*. Based on this evaluation, the meta-agent generates targeted feedback. The MAS, its intermediate outputs, and the feedback are stored in the *experience library*, which is then used as additional context to refine the design in subsequent iterations.

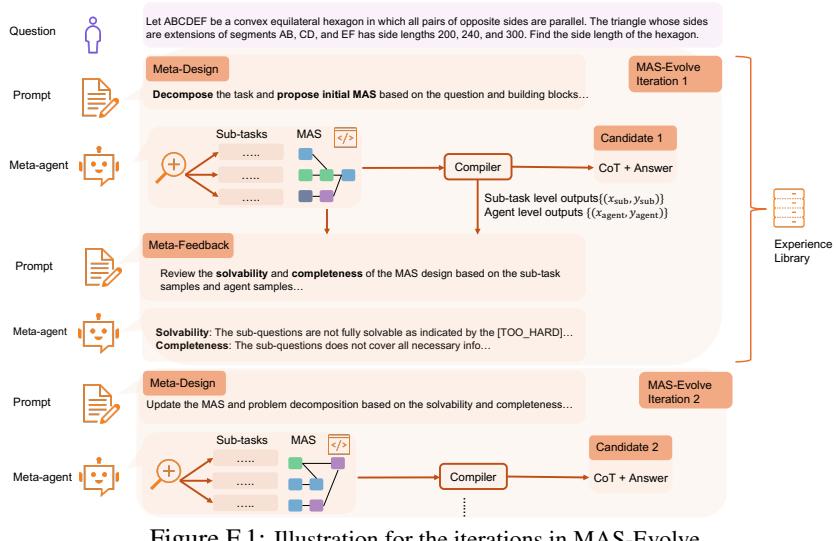


Figure F.1: Illustration for the iterations in MAS-Evolve.

G EXAMPLE OF MAS PRODUCED FROM MAS-ZERO

MAS-ZERO learns to decompose a new question and assign appropriate sub-MAS to each sub-task dynamically. This type of dynamic assignment would have been difficult to design manually. Fig. G.1 showcases the effectiveness of MAS-ZERO by demonstrating how it can construct and refine MAS architectures on the fly, adapting complexity to the requirements of the task.

H PROMPT DETAILS

As described in Sec. 3, MAS-ZERO uses prompts to implement its three steps. In this section, we present all detailed prompts used, including the building blocks (Fig. J.2, J.3, J.4, J.5) in MAS-Init;

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

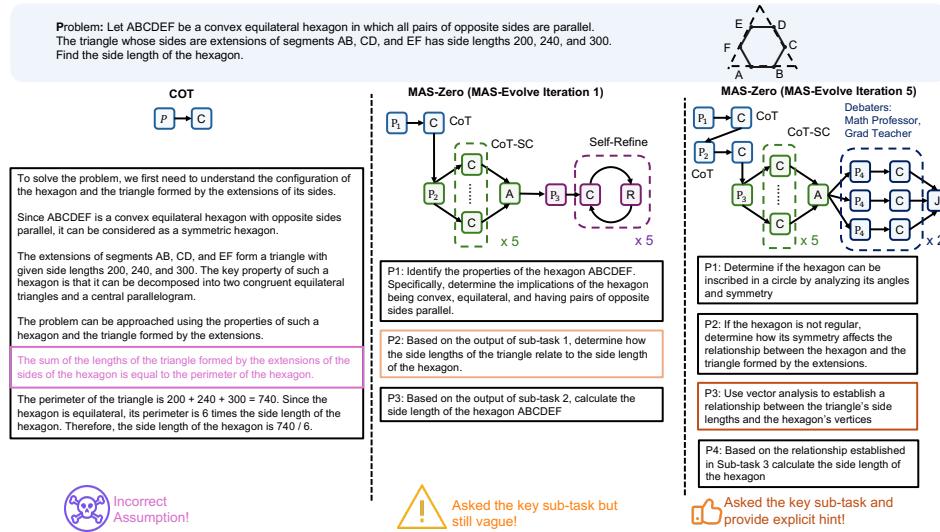


Figure G.1: An example illustrates how the MAS produced by MAS-ZERO outperforms both CoT and early iterations. In this case, MAS-ZERO learns to decompose the task into 4 sub-tasks and dynamically assign appropriate sub-MAS: CoT for the first two, CoT-SC (sampling 5 completions) for the third, and Debate (2 rounds with a math professor and a graduate teacher as debaters) for the fourth.

the Meta-Design (Fig. H.1) and Meta-Feedback (Fig. H.2) in MAS-Evolve; and the prompts for MAS-Verify (Fig. H.3).

I CODE TEMPLATE

MAS-ZERO uses a code template to aid MAS code generation to filling in a specific forward function. Fig. J.1 shows how the utility code is provided. Fig.s J.2, J.3, J.4, and J.5 show the implementations of each building blocks.

J USAGE OF LARGE LANGUAGE MODELS

We use LLMs solely for grammar polishing.

972
973
974
975
976
977
978
979
980

Prompt for Meta-Design

Overview. You are an expert machine learning researcher testing various agentic systems. You are given a set of building blocks (e.g., CoT, CoT-SC, Self-Refine, Debate) and a question. Each building block is represented as executable code and can contain one or more LLM agents with specialized settings (instruction, temperature, etc.). A *sub-MAS* refers to one or more building blocks assigned together to solve a specific sub-task. The overall MAS is formed by coordinating multiple sub-MAS to solve the full problem.

Your objectives are:

- **Perform task decomposition.** Decompose the given question into *manageable yet interdependent* sub-tasks, such that each sub-task is specific and detailed enough for a sub-MAS (formed from building blocks) to solve. Do not solve the tasks yourself or leak the expected answer. Instead, design the decomposition so that the sub-tasks are easier for sub-MAS to solve, and justify how they combine to yield the final answer.

Make sure

- Each sub-task should explicitly build on the outputs of prior sub-tasks.
- The final sub-task should naturally yield the overall answer to the original question.

- **Design sub-MAS assignments.** Based on the resulting sub-tasks, assign one or more building blocks to form a sub-MAS for each sub-task. You may adjust block parameters (e.g., temperature, number of debate rounds), but you must not invent new blocks or prune the provided ones without grounding.

For example: Given available building blocks {CoT, CoT-SC, Self-Refine, Debate}, and the resulting sub-tasks: sub-task 1, sub-task 2, sub-task 3:

- **Step 1: For each sub-task, specify its sub-MAS.**

- * It may use a **single block** (e.g., Sub-task 1: CoT).
- * It may use a **sequential chain of blocks** (e.g., Sub-task 3: CoT → Self-Refine).
- * Or it may use **parallel blocks inside the sub-task** (e.g., Sub-task 2: {CoT || Debate}), meaning both blocks process the same input).

- **Step 2: Connect the sub-tasks (sub-MAS).** After defining sub-MAS, specify how the sub-tasks depend on one another:

- * **Sequential connection:** Sub-tasks are connected in a linear chain, where the output of one becomes the input to the next.

Example: [CoT] (sub-task 1) → [Debate] (sub-task 2) → [CoT → Self-Refine] (sub-task 3).

- * **Parallel connection:** Multiple sub-tasks depend on the same earlier sub-task's output and run in parallel.

Example: [CoT] (sub-task 1) → {[Debate] (sub-task 2), [CoT → Self-Refine] (sub-task 3)}.

Here, both sub-task 2 and sub-task 3 consume the result of sub-task 1 in parallel.

IMPORTANT: Do not collapse all decomposed sub-tasks into a single instruction handled by one block. Each sub-task must be addressed by its own sub-MAS.

Final remark: Your aim is to design an optimal block connection that can perform well on each sub-task. Your code should implement the existing blocks as-is. Do not propose new blocks or modify existing ones; you may only adjust their connections and settings (e.g., instruction, temperature).

Figure H.1: Prompt for Meta-Design. Additional examples, building blocks code and output format instruction are omitted for clarity.

1017
1018
1019
1020
1021
1022
1023
1024
1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

Prompt for Meta-Feedback

Overview. You are given a candidate Multi-Agent System (MAS) design, including: (i) its executable code, (ii) the sub-task outputs from each sub-MAS, (iii) the outputs of individual agents, (iv) the final response, and (v) experience of prior iterations. Your task is to critically evaluate this MAS and provide feedback to guide refinement in the next iteration.

- **Solvability:** Assess whether all sub-tasks are solvable by the corresponding block via checking the output answer of each sub-task.

- If the output explicitly includes [TOO_HARD], this means the sub-task is too difficult and should be further decomposed.

- If the output is incorrect, identify whether the issue is due to

- Insufficient decomposition (the task is still too hard) or

- Some agents in the block are malfunctioning, or the underlying LLM is too weak to solve the sub-task. This can be checked by reviewing the agent outputs: (a) If the agent itself is not optimal (e.g., poor instruction, temperature, etc.), the settings need to be improved. (b) If the agent architecture is not optimal, a new block should be proposed by recombining existing blocks or adjusting their settings.

Please **justify** whether it is (i), the decomposition issue or (ii) the block and agent issue. It could also be both. When proposing new sub-task, **make sure**

- It is specific and detailed enough to solve and to contribute to the final answer;

- All required information is carried over from previous sub-tasks or provided in the instruction;

- The outputs from related sub-tasks are correctly incorporated (e.g., added to the taskInfo list when calling the agent);

- The connection to prior sub-tasks is explicit (e.g., instructions should state “Based on the output of sub-task i ”).

- **Completeness:** Do the sub-tasks include all necessary information from the original query that can ensure the aggregation of sub-task responses can effectively yield a comprehensive answer to the user query? Note that while a sub-task might include only part of the necessary information, it is not allowable for any particular piece of critical information to be omitted from all sub-tasks.

- If critical information is missing, refine the decomposition to include it.

- Ensure sub-tasks are connected so that the aggregated outputs can yield a correct and comprehensive final answer.

Now, you need to improve or revise the implementation, or implement the new proposed MAS based on the reflection.

Figure H.2: Prompt for Meta-Feedback. Additional examples, building blocks code and output format instruction are omitted for clarity.

Prompt for MAS-Verify

Given the problem and a list of candidate answers, carefully review the reasoning steps and final answers to select the most reliable candidate. Do not solve the task yourself.

In your output, use the “thinking” field to compare the selected answer with each unselected answer one by one, identify the erroneous steps in the unselected answers, and give a detailed explanation of why they are incorrect. In the “selection” field, output the ID of the best answer.

Problem: {problem}

Answer List: {candidate answers}

Figure H.3: Prompt for MAS-Verify. The “candidate answers” have already been ranked and filtered before being passed to the meta-agent. Additional examples and output format instructions are omitted for clarity.

1078

1079

```

1080
1081
1082
1083
1084
1085
1086
1087
1088 Utility Code
1089
1090
1091     # Named tuple for holding task information
1092     Info = namedtuple('Info', ['name', 'author', 'content', 'prompt', 'sub_tasks', 'agents'
1093                           , 'iteration_idx'])
1094
1095     # Format instructions for LLM response
1096     FORMAT_INST = lambda request_keys: f"Reply EXACTLY with the following JSON format.\n{{
1097         str(request_keys)}\nDO NOT MISS ANY FIELDS AND MAKE SURE THE JSON FORMAT IS
1098         CORRECT!\n"
1099
1100    # Description of the role for the LLM
1101    ROLE_DESC = lambda role: f"You are a {role}."
1102
1103    class LLMAgentBase():
1104
1105        def __init__(self, output_fields: list, agent_name: str,
1106                     role='helpful assistant', model=None, temperature=None) -> None:
1107            self.output_fields = output_fields
1108            self.agent_name = agent_name
1109
1110            self.role = role
1111            self.model = model
1112            self.temperature = temperature
1113            # give each instance a unique id
1114            self.id = random_id()
1115
1116
1117            def generate_prompt(self, input_infos, instruction) -> str:
1118                # generate prompt based on the input_infos
1119                # ...
1120
1121            def query(self, input_infos: list, instruction, iteration_idx=-1) -> dict:
1122                # call generate_prompt and the LLM to get output
1123                # ...
1124
1125            def __repr__(self):
1126                return f"{self.agent_name} {self.id}"
1127
1128
1129    class AgentArchitecture:
1130        """
1131            Fill in your code here.
1132
1133            def forward(self, taskInfo) -> Union[Info, str]:
1134                Args:
1135                    - taskInfo (Info): Task information.
1136
1137                    Returns:
1138                    - Answer (Info): Your FINAL Answer.
1139        """
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233

```

Figure J.1: Utility code.

1134

1135

1136 Implementation of CoT Building Blocks

1137

```

1138     def forward(self, taskInfo):
1139         # Instruction for the Chain-of-Thought (CoT) approach
1140         # It is an important practice that allows the LLM to think step by step before
1141             # solving the task.
1142         cot_instruction = self.cot_instruction
1143
1144         # Instantiate a new LLM agent specifically for CoT
1145         # To allow the LLM to think before answering, we need to set an additional output
1146             # field 'thinking'.
1147         cot_agent = LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent', model=
1148             self.node_model, temperature=0.0)
1149
1150         # Prepare the inputs for the CoT agent
1151         # The input should be a list of Info, and the first one is often the taskInfo
1152         cot_agent_inputs = [taskInfo]
1153
1154         # Get the response from the CoT agent
1155         thinking, answer = cot_agent(cot_agent_inputs, cot_instruction)
1156         final_answer = self.make_final_answer(thinking, answer)
1157
1158         # Return only the final answer
1159         return final_answer
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

```

Figure J.2: Implementation of CoT building blocks

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

Implementation of CoT-SC Building Blocks

```

def forward(self, taskInfo):
    # Instruction for step-by-step reasoning
    cot_instruction = self.cot_instruction
    N = self.max_sc # Number of CoT agents

    # Initialize multiple CoT agents with a higher temperature for varied reasoning
    cot_agents = [LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent', model=
        self.node_model, temperature=0.5) for _ in range(N)]

    # Majority voting function to select the most common answer
    from collections import Counter
    def majority_voting(answers):
        return Counter(answers).most_common(1)[0][0]

    thinking_mapping = {}
    answer_mapping = {}
    possible_answers = []
    for i in range(N):
        thinking, answer = cot_agents[i]([taskInfo], cot_instruction)
        possible_answers.append(answer.content)
        thinking_mapping[answer.content] = thinking
        answer_mapping[answer.content] = answer

    # Ensembling the answers from multiple CoT agents
    answer = majority_voting(possible_answers)

    thinking = thinking_mapping[answer]
    answer = answer_mapping[answer]

    final_answer = self.make_final_answer(thinking, answer)

    return final_answer

```

Figure J.3: Implementation of CoT-SC building blocks

1186

1187

```

1188
1189
1190
1191
1192
1193
1194
1195
1196
1197 Implementation of Debate Building Blocks
1198
1199
1200     def forward(self, taskInfo):
1201         # Instruction for initial reasoning
1202         debate_initial_instruction = self.cot_instruction
1203
1204         # Instruction for debating and updating the solution based on other agents'
1205         # solutions
1206         debate_instruction = "Given solutions to the problem from other agents, consider
1207         # their opinions as additional advice. Please think carefully and provide an
1208         # updated answer. Put your thinking process in the 'thinking' field and the
1209         # updated answer in the 'answer' field. "
1210
1211         # Initialize debate agents with different roles and a moderate temperature for
1212         # varied reasoning
1213         debate_agents = [LLMAgentBase(['thinking', 'answer'], 'Debate Agent', model=self.
1214             node_model, role=role, temperature=0.5) for role in self.debate_role]
1215
1216         # Instruction for final decision-making based on all debates and solutions
1217         final_decision_instruction = "Given all the above thinking and answers, reason over
1218         # them carefully and provide a final answer. Put your thinking process in the 'thinking'
1219         # field and the final answer in the 'answer' field."
1220         final_decision_agent = LLMAgentBase(['thinking', 'answer'], 'Final Decision Agent',
1221             model=self.node_model, temperature=0.0)
1222
1223         max_round = self.max_round # Maximum number of debate rounds
1224         all_thinking = [[] for _ in range(max_round)]
1225         all_answer = [[] for _ in range(max_round)]
1226
1227         # Perform debate rounds
1228         for r in range(max_round):
1229             for i in range(len(debate_agents)):
1230                 if r == 0:
1231                     thinking, answer = debate_agents[i]([taskInfo],
1232                         debate_initial_instruction)
1233                 else:
1234                     input_infos = [taskInfo] + [all_thinking[r-1][i]] + all_thinking[r-1][:
1235                         i] + all_thinking[r-1][i+1:]
1236                     thinking, answer = debate_agents[i](input_infos, debate_instruction)
1237                     all_thinking[r].append(thinking)
1238                     all_answer[r].append(answer)
1239
1240         # Make the final decision based on all debate results and solutions
1241         thinking, answer = final_decision_agent([taskInfo] + all_thinking[max_round-1] +
1242             all_answer[max_round-1], final_decision_instruction)
1243         final_answer = self.make_final_answer(thinking, answer)
1244
1245         return final_answer
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341

```

Figure J.4: Implementation of Debate building blocks

```

1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254 Implementation of Self-Refine Building Blocks
1255
1256     def forward(self, taskInfo):
1257         # Instruction for initial reasoning
1258         cot_initial_instruction = self.cot_instruction
1259
1260         # Instruction for reflecting on previous attempts and feedback to improve
1261         cot_reflect_instruction = "Given previous attempts and feedback, carefully consider
1262             where you could go wrong in your latest attempt. Using insights from previous
1263             attempts, try to solve the task better."
1264         cot_agent = LLMAgentBase(['thinking', 'answer'], 'Chain-of-Thought Agent', model=
1265             self.node_model, temperature=0.0)
1266
1267         # Instruction for providing feedback and correcting the answer
1268         critic_instruction = "Please review the answer above and criticize on where might
1269             be wrong. If you are absolutely sure it is correct, output exactly 'True' in '"
1270             "correct'."
1271         critic_agent = LLMAgentBase(['feedback', 'correct'], 'Critic Agent', model=self.
1272             node_model, temperature=0.0)
1273
1274         N_max = self.max_round # Maximum number of attempts
1275
1276         # Initial attempt
1277         cot_inputs = [taskInfo]
1278         thinking, answer = cot_agent(cot_inputs, cot_initial_instruction, 0)
1279
1280         for i in range(N_max):
1281             # Get feedback and correct status from the critic
1282             feedback, correct = critic_agent([taskInfo, thinking, answer],
1283                 critic_instruction, i)
1284             if correct.content == 'True':
1285                 break
1286
1287             # Add feedback to the inputs for the next iteration
1288             cot_inputs.extend([thinking, answer, feedback])
1289
1290             # Reflect on previous attempts and refine the answer
1291             thinking, answer = cot_agent(cot_inputs, cot_reflect_instruction, i + 1)
1292
1293         final_answer = self.make_final_answer(thinking, answer)
1294
1295         return final_answer

```

Figure J.5: Implementation of Self-Refine building blocks

```

1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

```