MASCRAFT: MULTI-AGENT SPECIALIZATION THROUGH ROLE–SKILL COMPOSITION AND SELF-EVOLVING ORCHESTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Effective multi-agent systems (MAS) require agents specialized for different types of work. However, role prompting primarily specifies who an agent should act as, without necessarily providing sufficient procedural guidance on how to perform its work. We introduce **MASCraft**, a framework that combines role–skill composition with self-evolving orchestration for multi-agent specialization. By pairing independently defined roles with reusable skills, MASCraft equips specialists with both professional identities and explicit procedural guidance. An orchestrator agent plans subtasks, selects role–skill pairs from separate libraries, and organizes collaboration according to information dependencies. To better meet the specialized demands of MAS orchestration, we provide an explicit policy that evolves through execution experience. This policy comprises two textual modules: *Environment Knowledge*, which records task characteristics and recurring failures, and *Team Strategy*, which guides specialist configuration and coordination. MASCraft generates multiple rollouts per task, selects tasks with contrasting results for within-task reflection, and aggregates lessons across tasks to update the policy, guiding subsequent team construction. Extensive experiments on HLE, DeepResearch Bench, and BrowseComp-Plus show that MASCraft consistently outperforms competing MAS methods across different LLM backbones. Code is included in the supplementary material.

1 INTRODUCTION

Advances in large language models (LLMs) (DeepSeek-AI, 2026; Qiu et al., 2026; Kimi Team, 2026) and harness engineering (Lopopolo, 2026; Young, 2025) have substantially expanded the capabilities of LLM-based agents (Yao et al., 2023; Chen et al., 2024b; Qian et al., 2024) across a wide range of domains, including mathematical reasoning (Gou et al., 2024), software engineering (Yang et al., 2024), and scientific research (Lu et al., 2026). Building on these advances, multi-agent systems (MAS) provide a promising paradigm for tackling complex tasks by scaling individual capabilities into collective intelligence (Guo et al., 2024; Dang et al., 2025; Ke et al., 2026; Kimi Team, 2026). Specifically, MAS coordinate multiple LLM-based agents to handle different types of work within a complex task, such as product design and algorithm implementation in AI application development (Hong et al., 2024; Wu et al., 2024). As these forms of work typically require distinct expertise, effective multi-agent collaboration naturally calls for *multi-agent specialization*: tailoring prompts to guide individual agents to act as specialists suited to their assigned work (Chen et al., 2024a; Vu et al., 2026).

Existing approaches to MAS construction commonly achieve multi-agent specialization through role prompting, specifying details such as agents' professional identities and communication styles (Li et al., 2023; Wu et al., 2024). These specialized agents form the basis of MAS, with teams either predefined—as in AI application development with product manager and algorithm engineer roles (Qian et al., 2024; Hong et al., 2024)—or assembled dynamically based on the task context (Chen et al., 2024a; Hu et al., 2026a; Xu et al., 2026; Lin et al., 2026). However, role prompting primarily specifies *who to act as*, without necessarily providing sufficient procedural guidance on *how to do the work* (Figure 1). For example, the prompt "You are an algorithm engineer specializing in CUDA kernel development" assigns a specialist identity without detailing how to implement and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

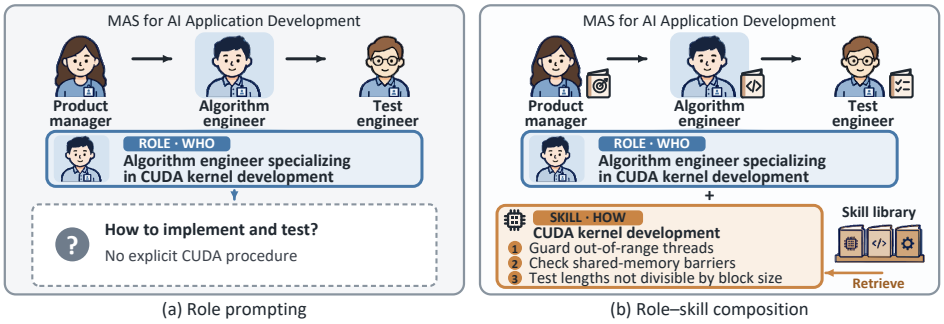


Figure 1: Role prompting versus role–skill composition in MAS for AI application development. Both teams retain the same specialist roles, while role–skill composition adds explicit procedural guidance, illustrated by a CUDA-specific skill for the algorithm engineer.

test CUDA kernels, potentially yielding plausible but functionally incorrect code (Zheng et al., 2024; Hu et al., 2026b; Luz de Araujo et al., 2025). This gap in procedural guidance can be amplified as MAS scale, with individual errors propagating and compounding through inter-agent interactions, ultimately undermining system reliability (Kim et al., 2025; Cemri et al., 2025).

To address this gap, MAS construction should equip agents with explicit procedural guidance alongside their assigned roles. This guidance should be composable, enabling the same role to draw on different procedures as its assigned work changes, and reusable, retaining established procedures rather than relying on potentially inconsistent on-the-fly generation (Belov et al., 2026). The emerging "skill" paradigm offers a natural solution by capturing procedural knowledge as step-by-step instructions and usage examples in structured Markdown documents (`SKILL.md`) (Xu & Yan, 2026; Li et al., 2026). These skills can then be collected into a library, from which they can be retrieved and flexibly paired with independently defined roles, supporting reusability and composability.

We next consider how to construct task-specific MAS based on role–skill composition. To this end, we use an orchestrator agent to plan the subtasks, select a role–skill pair from the respective libraries to configure a specialist agent for each subtask, and organize collaboration among specialists according to their information dependencies. The specialist agents then execute their subtasks in dependency order, following their assigned role and skill instructions and incorporating upstream outputs to produce the final answer. Overall, this approach supports more comprehensive multi-agent specialization, addressing not only *who to act as* but also *how to do the work*.

However, existing agents may struggle with the specialized demands of MAS orchestration without targeted training (Ke et al., 2026; Estornell et al., 2025; Nielsen et al., 2026), motivating an explicit orchestration policy to guide team construction under our role–skill formulation. Specifying an effective policy upfront is challenging: key task characteristics may emerge only through exploration, while the effectiveness of team configurations is difficult to assess before execution. The policy should therefore evolve with execution experience. To this end, we maintain two evolving textual modules: *Environment Knowledge*, which records task characteristics and recurring failures, and *Team Strategy*, which captures lessons for configuring and coordinating specialists. We generate multiple rollouts per task, select tasks with contrasting results for within-task reflection, and aggregate lessons across tasks to update both modules and guide subsequent team construction. Together, role–skill composition and self-evolving orchestration form our framework, **MASCraft**. Experiments on HLE (Center for AI Safety et al., 2026), DeepResearch Bench (Du et al., 2026), and BrowseComp-Plus (Chen et al., 2026) demonstrate consistent gains over competing MAS methods across all three LLM backbones.

The main contributions of this work are summarized as follows:

- We propose a role–skill formulation of multi-agent specialization that complements role definitions with reusable procedural guidance, addressing both *who to act as* and *how to do the work*.
- We introduce MASCraft, a framework that uses an orchestrator agent to construct task-specific MAS through role–skill composition and refines its orchestration policy through self-evolution from execution experience.

- We conduct extensive experiments on three agent benchmarks with three LLM backbones, demonstrating consistent performance gains over competing MAS methods.

2 METHODOLOGY

In this section, we first introduce MASCraft, then detail its two components: role–skill composition and self-evolving orchestration.

2.1 OVERVIEW

We aim to specialize agents by explicitly specifying both *who to act as* and *how to do the work*, while improving how these specialists are organized for a given task. To achieve this, our key idea is to complement role definitions with procedural guidance from reusable skills and use execution experience to improve team construction.

Building on this idea, MASCraft uses an LLM-based orchestrator agent that plans and organizes specialists' work to construct task-specific MAS (Figure 2). MASCraft integrates two complementary components: (1) *Role–Skill Composition*, through which the orchestrator agent plans subtasks, selects role–skill pairs to configure specialists, and establishes their information dependencies; and (2) *Self-Evolving Orchestration*, which refines a natural-language orchestration policy through within-task reflection and cross-task aggregation to guide subsequent team construction.

2.2 ROLE–SKILL COMPOSITION

To specify both who each specialist should act as and how it should perform its assigned work, the orchestrator agent pairs a role with a skill for each subtask. We describe the libraries, task-specific orchestration, and execution below.

2.2.1 ROLE AND SKILL LIBRARIES

To support reusable instructions and flexible role–skill pairing, we maintain separate libraries for roles, $\mathcal{R}$, and skills, $\mathcal{S}$. This allows the same role to be paired with different procedures as its assigned work changes. The libraries can draw on public or organization-specific resources, such as job descriptions for roles and standard operating procedures or expert practices for skills. Our library construction procedure is detailed in Appendix B.3.

Each entry is a structured natural-language document containing a name, a short description, and a full instruction body. The description summarizes when the entry is useful. A role's body specifies responsibilities and a professional perspective, while a skill's body provides procedural guidance, including applicable conditions, steps, and checks (Xu & Yan, 2026; Li et al., 2026). Descriptions support selection, while full instruction bodies guide specialist execution.

2.2.2 MULTI-AGENT ORCHESTRATION

To construct a task-specific MAS, the orchestrator agent jointly plans subtasks, selects role–skill pairs from the libraries to configure specialists, and establishes their information dependencies. In our implementation, the orchestrator agent generates a team plan from the original task x, library entry descriptions, and policy π (expressed in natural language; see Section 2.3):

$$G = \mathrm{Orchestrator}(x, \mathrm{desc}(\mathcal{R}), \mathrm{desc}(\mathcal{S}), \pi)\,, \tag{1}$$

where $\mathrm{Orchestrator}$ denotes the orchestrator agent and desc returns library entry names and short descriptions. The generated plan is $G = (\{(q_i, r_i, s_i)\}_{i=1}^{n}, E)$, where n is the number of specialist agents and each triple specifies a subtask q_i, a role $r_i \in \mathcal{R}$, and a skill $s_i \in \mathcal{S}$. The edge set E defines information dependencies: $(j, i) \in E$ means that specialist agent i requires specialist agent j's output. We index the final specialist agent as n; it receives all other specialists' outputs to produce the final answer. These dependencies form a directed acyclic graph (DAG). More generally, role–skill selection is implementation-agnostic and can incorporate retrieval and reranking to limit context overhead when the libraries grow large.

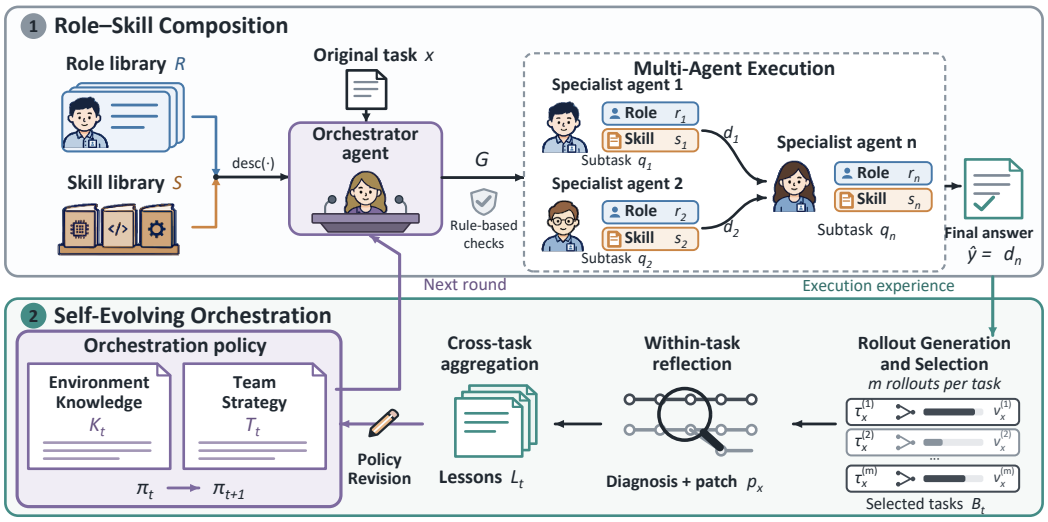


Figure 2: Overview of MASCraft. (1) *Role–Skill Composition*: the orchestrator agent assigns subtasks, selects role–skill pairs, and establishes information dependencies. (2) *Self-Evolving Orchestration*: within-task reflection and cross-task aggregation distill lessons from contrasting rollouts to update *Environment Knowledge* and *Team Strategy* for subsequent team construction.

Before execution, rule-based checks verify the selected library entries, valid and acyclic dependencies, and a unique final specialist receiving all other outputs. Failed checks return feedback to the orchestrator agent for revision and revalidation. This stage yields a validated team plan.

2.2.3 Multi-Agent Execution

To execute the fixed team plan G, each specialist receives its full role and skill instructions, the complete original task x, and its subtask q_i. Keeping x visible preserves the overall requirements during subtask execution. Once all required upstream outputs are available, specialist agent i produces d_i:

$$d_i = \text{Specialist}(x, q_i, \text{body}(r_i), \text{body}(s_i), \{d_j : (j,i) \in E\}), \tag{2}$$

where body retrieves an entry's full instructions and Specialist denotes execution by the configured specialist agent using these inputs and any permitted tools. Independent agents can run concurrently.

The final specialist agent n integrates $d_1, \ldots, d_{n-1}$ under its own role and skill instructions to produce the answer $\hat{y} = d_n$. The resulting rollouts inform subsequent policy updates.

2.3 Self-Evolving Orchestration

General-purpose LLMs may struggle with MAS orchestration due to limited targeted training in task delegation and multi-agent coordination (Ke et al., 2026; Estornell et al., 2025; Nielsen et al., 2026). We therefore guide the orchestrator agent with an explicit natural-language orchestration policy π. Execution exposes task difficulties and indicates which configurations work well, providing lessons for policy refinement. We detail the formulation and update procedure below.

2.3.1 Policy Formulation

To guide team construction, the policy must capture task requirements and how specialists should address them. We therefore represent it at round t as two complementary textual modules, $\pi_t = (K_t, T_t)$: *Environment Knowledge*, K_t, records task characteristics and recurring failures, while *Team Strategy*, T_t, guides specialist configuration and coordination based on this understanding. Both modules are supplied to the orchestrator agent to generate G in Equation 1.

Each policy module is a structured Markdown document with four fixed sections. Environment Knowledge contains *Task Distribution*, *Evidence Landscape*, *Inference Characteristics*, and *Ob-*

served Failure Modes; Team Strategy contains *Task Decomposition*, *Role–Skill Composition*, *Dependency Design*, and *Team Quality Checks*. Examples appear in Appendix A.3. Updates revise the full documents under fixed headings, adding, removing, or consolidating guidance based on execution experience.

2.3.2 POLICY UPDATE

Rollout Generation and Selection. At round t, we generate m rollouts per validation task x under π_t to compare alternative executions. The kth rollout is $\tau_x^{(k)}$, with evaluation score $v_x^{(k)}$:

$$\tau_x^{(k)} = \mathrm{Rollout}(x; \pi_t), \qquad v_x^{(k)} = \mathrm{Eval}_x\left(\tau_x^{(k)}\right), \quad k = 1, \ldots, m, \tag{3}$$

Here, $\mathrm{Rollout}$ denotes a complete run of Equations 1–2; its record contains G, specialist outputs, and $\hat{y}$. The task-specific evaluator Eval_x scores the final answer. To provide contrasts for reflection, we select tasks using the mean-score gap between their two highest- and two lowest-scoring rollouts, requiring a positive gap for binary correctness or a preset minimum gap for continuous scores. Let $\mathcal{B}_t$ denote the selected tasks. The round index is omitted from rollout and task-level notation.

Contrastive Reflection and Aggregation. We analyze execution experience at two levels: within-task reflection examines differences between rollouts, while cross-task aggregation extracts lessons beyond individual tasks. For each selected task $x \in \mathcal{B}_t$, an LLM reflector locates divergences in how rollouts address the same requirements and traces their potential effects on later results. The resulting analysis p_x contains a diagnosis of what went wrong and a targeted patch suggesting how to improve future orchestration. An LLM then aggregates these analyses by grouping recurring problems, retaining supported practices, and merging overlapping remedies into lessons L_t:

$$\begin{aligned} p_x &= \mathrm{Reflect}\left(x, \pi_t, \{(\tau_x^{(k)}, v_x^{(k)})\}_{k=1}^{m}\right), \\ L_t &= \mathrm{Aggregate}(\pi_t, \{(x, p_x)\}_{x \in \mathcal{B}_t}), \end{aligned} \tag{4}$$

Both LLM-based operations use π_t to assess proposed changes against existing guidance. Reflection also uses available reference answers or grading feedback.

Policy Revision. An LLM writer incorporates the aggregated lessons L_t into the current policy:

$$\pi_{t+1} = \mathrm{Update}(\pi_t, L_t), \tag{5}$$

where Update revises Environment Knowledge and Team Strategy, retaining supported guidance and excluding task identifiers or instance-specific solutions. A module remains unchanged when no revision is justified. The revised policy guides subsequent team construction using these lessons.

3 EXPERIMENTS

In this section, we conduct a series of experiments to answer the following research questions:

- **RQ1:** How does MASCraft perform compared to existing LLM-based agent methods?
- **RQ2:** What is the contribution of each component within MASCraft to the overall performance?
- **RQ3:** How does the orchestration policy evolve to improve team construction?

3.1 EXPERIMENTAL SETUP

Benchmarks and Evaluation Metrics. We evaluate MASCraft on three public benchmarks: the mathematics subset of **Humanity's Last Exam (HLE)** (Center for AI Safety et al., 2026) for mathematical reasoning, **DeepResearch Bench (DRB)** (Du et al., 2026) for deep research tasks, and **BrowseComp-Plus (BCP)** (Chen et al., 2026) for complex multi-hop question answering. We report accuracy for HLE and BCP. For DRB, we assess comprehensiveness, insight, instruction following, and readability using the overall score from RACE (Reference-based Adaptive Criteria-driven Evaluation), which combines task-specific weighting with reference-based scoring (Du et al., 2026). Validation/test sizes are 32/168 for HLE and BCP and 16/84 for DRB; scores average all test examples. All methods share the same splits and tool settings: HLE and DRB use no external tools, while BCP uses local BM25 retrieval. Sampling and evaluation details appear in Appendix B.1.

Table 1: Overall performance across three LLM backbones. HLE denotes the mathematics subset of Humanity's Last Exam. Scores are accuracy (%) on HLE and BCP and RACE overall scores ($\times 100$) on DRB. Bold and underlined values indicate the best and second-best results in each column, respectively; ties are retained. Higher is better.

Method	DeepSeek-V4-flash			Qwen3.8-flash			GLM-5.2-fast-preview		
	HLE	DRB	BCP	HLE	DRB	BCP	HLE	DRB	BCP
Single	10.71	44.73	11.31	14.29	45.58	13.10	11.90	45.41	17.86
Self-Refine	9.52	46.93	13.10	19.05	46.62	14.88	11.31	44.85	16.67
MAD	10.71	44.47	12.50	20.83	46.80	20.83	16.07	44.86	17.86
AutoAgents	13.10	48.96	22.62	16.07	46.13	21.43	14.88	46.85	25.00
AFlow	10.12	<u>49.56</u>	19.64	20.24	47.58	20.83	10.12	44.10	16.07
EvoMAS	13.69	49.04	17.86	22.02	<u>49.14</u>	22.62	<u>17.86</u>	46.60	25.00
Meta-Team	11.90	45.98	17.26	17.86	45.91	27.38	11.31	44.36	28.57
TacoMAS	10.71	49.43	<u>23.21</u>	<u>22.62</u>	46.92	25.60	8.93	50.07	30.36
Skill-MAS	<u>20.24</u>	47.68	<u>23.21</u>	<u>22.62</u>	48.55	<u>27.98</u>	15.48	<u>50.49</u>	<u>35.12</u>
MASCraft	**26.19**	**50.78**	**29.76**	**26.19**	**49.35**	**37.50**	**20.24**	**52.21**	**38.10**

Baselines. We compare MASCraft with three groups of baselines: (1) single-agent methods, **Single** and **Self-Refine** (Madaan et al., 2023); (2) static MAS methods without cross-task evolution, **MAD** (Du et al., 2024) and **AutoAgents** (Chen et al., 2024a); and (3) MAS methods with self-evolution, **AFlow** (Zhang et al., 2025), **EvoMAS** (Hu et al., 2026a), **Meta-Team** (Hao et al., 2026), **TacoMAS** (Xu et al., 2026), and **Skill-MAS** (Lin et al., 2026). Self-evolving baselines use the same validation split and freeze the selected configurations before testing. Further configurations are detailed in Appendix B.2.

LLM Backbones. We use **DeepSeek-V4-flash**, **Qwen3.8-flash**, and **GLM-5.2-fast-preview** as backbones for all methods. During team execution and policy evolution, the same backbone serves the orchestrator agent, specialist agents, reflection, aggregation, and policy revision; only the LLM judge uses **GPT-5-mini-0807**. All model parameters remain fixed.

Implementation Settings. We construct role and skill libraries by prompting **GPT-5.6-sol** to extract task patterns from validation and other available non-test tasks, generate entries with manually specified fields, and deduplicate and merge them. The libraries are manually reviewed and remain fixed during evolution. The same model generates initial policies from ten validation examples per benchmark and background on team construction and execution. MASCraft forms teams of two to five specialists. We evaluate ten policy rounds, including initialization, with five rollouts per validation task. The policy with the best validation score is frozen for one rollout per test task. Further details appear in Appendices B.3 and B.4.

3.2 Overall Performance (RQ1)

We begin by assessing the overall task performance of the compared methods across all three benchmarks and LLM backbones. Table 1 summarizes the results, yielding the following key observations:

- **MASCraft consistently outperforms the baselines.** Across all nine backbone–benchmark settings, it achieves a mean relative improvement of 15.06% over the strongest baseline in each setting. This advantage supports combining role–skill composition for explicit procedural guidance with self-evolving orchestration that refines specialist configuration and coordination.
- **MAS generally outperforms single-agent methods.** Compared with Single, Self-Refine yields less consistent performance gains than MAS methods. These results suggest that MAS benefits extend beyond increased inference effort, with multi-agent specialization enabling complementary work to be assigned to different agents.
- **Self-evolution generally improves MAS performance.** Self-evolving MAS methods generally outperform static baselines such as AutoAgents and MAD. This advantage may stem from iter-

Table 2: Ablations and baseline augmentation with GLM-5.2-fast-preview. Metrics follow Table 1. Random Skill samples one skill per specialist from the same library. K/T: Environment Knowledge/Team Strategy; $0/\star$: initial/evolved. +R–S adds manually selected role–skill pairs.

(a) Role–Skill Composition

Variant	HLE	DRB	BCP
MASCraft	20.24	52.21	38.10
Random Skill	13.69	51.63	33.33
Random Role	19.05	51.41	37.50
w/o Skill	16.07	51.08	33.93
w/o Role	16.67	51.59	35.71
w/o Both	11.31	50.79	32.74

(b) Self-Evolving Orchestration

Variant	HLE	DRB	BCP
MASCraft	20.24	52.21	38.10
Staged Plan	17.86	52.04	36.31
$(K_0, T^\star)$	17.26	51.55	36.31
$(K^\star, T_0)$	17.26	51.36	36.90
(K_0, T_0)	14.88	51.06	34.52
$(\varnothing, \varnothing)$	11.90	50.92	33.33

(c) Baseline Augmentation

Method	HLE		DRB		BCP	
	Base	+R–S	Base	+R–S	Base	+R–S
Single	11.90	13.10	45.41	47.54	17.86	19.64
Skill-MAS	15.48	19.05	50.49	51.70	35.12	36.31

atively improving agent behavior and collaboration through execution feedback. These results support evolving our orchestration policy to improve specialist configuration and coordination.

3.3 Ablation Studies (RQ2)

We assess MASCraft's components and role–skill guidance for existing methods through three groups of variants, using **GLM-5.2-fast-preview** under the main evaluation settings. Appendix C.2 collects the additional ablation results.

- **Role–Skill Composition.** We remove role or skill interfaces and their associated guidance (**w/o Role**, **w/o Skill**, and **w/o Both**), or assign roles or skills randomly from the same libraries after planning (**Random Role** and **Random Skill**). Each variant constructs a fresh team, with the tools and evidence budgets unchanged.
- **Self-Evolving Orchestration. Staged Plan** fixes subtasks and dependencies before selecting role–skill pairs under the same policy and libraries. We reset Environment Knowledge K, Team Strategy T, or both at inference: $(K_0, T^\star)$, $(K^\star, T_0)$, and (K_0, T_0) (0: initial; $\star$: evolved). $(\varnothing, \varnothing)$ removes both modules, retaining the base planning prompt and rule-based checks.
- **Baseline Augmentation.** We augment Single and the most competitive baseline, Skill-MAS, with manually selected role–skill pairs (**+R–S**) from the same libraries, using MASCraft's selections as a reference while preserving the baselines' execution and orchestration.

The summarized results are presented in Table 2, yielding the following observations:

- **Roles and skills provide complementary guidance.** Removing either component lowers scores; removing both causes the largest decline. Randomly selecting roles or skills also reduces performance, highlighting the need to jointly select both according to task requirements.
- **Self-evolution improves task performance.** Restoring either Environment Knowledge or Team Strategy to its initial version lowers scores; restoring both causes a larger decline. These results support evolving both modules through execution feedback.
- **Role–skill guidance transfers across methods.** Adding role–skill pairs improves Single and Skill-MAS, but both still trail MASCraft. This remaining gap may arise because neither framework is specifically designed to integrate role–skill composition into team construction or to refine this orchestration through self-evolution.

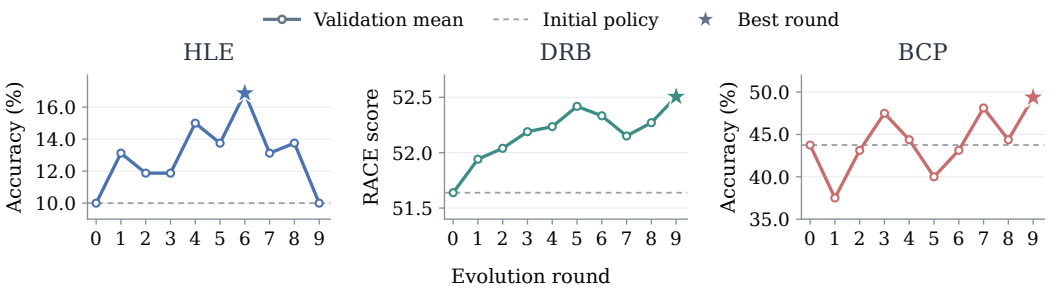


Figure 3: Validation performance across ten policy rounds with GLM-5.2-fast-preview, including initialization at round 0. Each point averages five rollouts per validation task. Dashed lines show initial-policy scores; stars mark the highest-scoring round for each benchmark in these runs.

Initial guidance	Common changes in final policies	Why this helps
Understand the task Describe typical tasks and likely mistakes.	**Explain which task conditions matter and what can go wrong in each case.** Use these differences to decide what work is needed.	**Address the actual difficulty, not just the topic of the task.**
Organize the work Divide the work and connect related tasks.	**For each task, specify who does it, what they need, and who uses their result.** Give each important step an owner and the required inputs.	**Avoid missing work or asking an agent to use results it lacks.**
Check the result Provide evidence and check the results.	**Specify what each agent must produce and how to check it before others use it.** Name the needed evidence and the checks to perform.	**Know whether an agent has finished its work before relying on it.**
Handle uncertainty State uncertainty and investigate the gaps.	**Say what to do when a result is missing, incomplete, or conflicts with other results.** Plan the next step instead of only noting the problem.	**Help later agents continue without guessing or giving up.**

Figure 4: Common changes across nine initial–final policy comparisons (three benchmarks and three backbones). Each row shows what becomes more explicit and how it guides team planning.

3.4 Self-Evolution Analysis (RQ3)

To examine how policy evolution affects validation performance and orchestration guidance, we first track ten GLM-5.2-fast-preview policy rounds. We then compare the initial and final policies used for the nine benchmark–backbone results in Table 1. We use GPT-5.6-sol to analyze changes in Environment Knowledge and Team Strategy, followed by manual verification, to identify common patterns across benchmarks and backbones.

Figure 3 shows that all three benchmarks achieve higher validation scores than initialization in their best rounds, although scores fluctuate during evolution. Comparing the initial and final policies reveals four common changes across benchmarks and backbones (Figure 4). For *task understanding*, the policies describe more clearly which task conditions matter and what can go wrong. For *work organization*, they specify how to divide the work, which inputs each agent needs, and who should make each key judgment. For *outputs and verification*, they go beyond asking agents to provide evidence and check their work: they specify what agents should produce, how to check it, and when the result is ready to use. For *handling incomplete information*, they explain what to do when evidence is missing or results disagree, such as recomputing disputed values, stating conclusions under explicit assumptions, or gathering more evidence. Full setting-level results and paired policy excerpts appear in Appendix C.3. These changes show how self-evolution turns execution experience into more concrete MAS guidance.

4 Related Work

Multi-Agent Specialization. Multi-agent specialization assigns different kinds of work to agents with suitable expertise. Role prompting describes agents' professional identities, responsibilities, and communication styles, helping establish a division of labor (Li et al., 2023; Wu et al., 2024;

Qian et al., 2024). However, specifying who an agent should act as does not necessarily explain how to perform its assigned work. Procedural guidance addresses this need through standard operating procedures embedded in collaborative workflows (Hong et al., 2024), or reusable workflows and skills that agents can apply during execution (Wang et al., 2025; Xu & Yan, 2026; Li et al., 2026). These approaches make the steps for carrying out a task explicit. Studies of persona prompting further show that expert identities alone do not consistently improve performance (Zheng et al., 2024; Luz de Araujo et al., 2025). Skills have also been studied as a way to capture multiple agents' behaviors within a single agent (Li, 2026). MASCraft pairs independently defined roles with reusable skills to specify both *who to act as* and *how to do the work*.

Automatic MAS Construction. Automatic MAS construction determines which agents to use, what work to assign them, and how their outputs should be combined. One line of work searches for reusable system designs, optimizing executable workflows or recombining functional modules (Hu et al., 2025; Zhang et al., 2025; Shang et al., 2025). Another configures teams for individual queries by generating agents and plans (Chen et al., 2024a;b), while evolutionary approaches refine structured team configurations from execution feedback (Hu et al., 2026a). Skill-aware routing also treats the choice of execution procedures and models as part of coordination (Koenigstein, 2026). These approaches differ in what is reused and what is decided for each task. A reusable workflow specifies how operations are organized, while an agent configuration specifies the participating agents and their instructions. A skill supplies a procedure that can be used within different assignments. This distinction affects how much of a system must be redesigned when the task changes. MASCraft jointly plans subtasks, selects role–skill pairs from separate libraries, and defines information dependencies to construct task-specific teams.

Self-Evolving MAS. Execution experience can improve a MAS by changing how agents perform their work or how they are organized. Some approaches train orchestrator models (Ke et al., 2026; Nielsen et al., 2026), while reflection and external memory retain lessons without updating LLM weights (Shinn et al., 2023). Within self-evolving MAS, methods differ in what they revise: specialists' accumulated expertise (Vu et al., 2026), reusable team behaviors and coordination rules (Hao et al., 2026), or agent capabilities and communication topology during task execution (Xu et al., 2026). This distinction separates improvements to individual agents from changes in how a team is constructed and coordinated. The former changes what agents know or how they work; the latter changes who is assigned to each subtask and which results are passed between agents. Skill-MAS is particularly close, using contrastive reflection over multiple rollouts to evolve textual guidance for MAS generation (Lin et al., 2026). MASCraft uses within-task reflection and cross-task aggregation to evolve an explicit policy for constructing teams through role–skill composition.

5 Conclusion

In this work, we presented **MASCraft**, a framework that combined role–skill composition with self-evolving orchestration for multi-agent specialization. We explicitly complemented role definitions with reusable procedural guidance, addressing not only *who to act as* but also *how to do the work*. Separate role and skill libraries allowed the orchestrator to configure specialists according to their assigned work. We further developed an evolving orchestration policy through reflection on contrasting rollouts within tasks and aggregation of lessons across tasks. We used these lessons to update *Environment Knowledge* and *Team Strategy*, enabling the orchestrator to account for task characteristics and team performance in subsequent team construction.

Building on this framework, future work will explore automatically expanding the role and skill libraries from validated execution experience. For larger skill libraries, we will investigate assigning each specialist a tailored skill pool with activation guided by its execution context. We will also explore dynamically selecting each agent's base model according to its assigned task, balancing task performance with inference cost and latency. Further directions include adapting team plans to runtime feedback, strengthening policy refinement under sparse or noisy feedback, and evaluating MASCraft on long-horizon interactive tasks involving multiple tools.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript drafting, translation, and revision, literature research, feedback on methodology and experimental design, interpretation of results, and figure preparation. GPT-5.6-sol generated the role and skill libraries from validation and other available non-test tasks, following the construction and manual review procedure in Appendix B.3. It also generated initial policies from ten validation examples per benchmark and background on team construction and execution (Appendix A.3). LLM use within agent execution, policy evolution, and evaluation is documented in the method and experimental sections and the appendix. The authors reviewed the AI-assisted text and figures, checked references and literature summaries against original sources, and reviewed the generated libraries for correctness and potential safety or ethical issues. We take full responsibility for the final content, claims, and artifacts of this work.

REFERENCES

Anthropic. Manage costs effectively. Claude Code Documentation, 2026.

Vitalii Belov, Artyom Sosedka, Andrey Sakhovskiy, Elizaveta Kovtun, Artyom Boyarskikh, and Semen Budennyy. LLM Agents Factory: Retrieval of domain-specific LLM agents. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 4474–4479, 2026.

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. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649(8099):1139–1146, 2026.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. AutoAgents: A framework for automatic agent generation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 22–30, 2024a.

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. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations (ICLR)*, 2024b.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A fair and disentangled evaluation benchmark for deep search agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 22349–22370, 2026.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, DeepSeek-AI, 2026.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Licheng Zhang, Xiaorui Wang, and Zhendong Mao. DeepResearch Bench: A comprehensive benchmark for deep research agents. In *International Conference on Learning Representations (ICLR)*, 2026.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)*, volume 235, pp. 11733–11763, 2024.

Andrew Estornell, Jean-Francois Ton, Muhammad Faaiz Taufiq, and Hang Li. How to train a leader: Hierarchical reasoning in multi-agent LLMs. *arXiv preprint arXiv:2507.08960*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations (ICLR)*, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 8048–8057, 2024.

Zhezheng Hao, Tianfu Wang, Huanshuo Dong, Ziyan Liu, Hong Wang, Xiankun Lin, Qiang Lin, Can Wang, Hande Dong, and Jiawei Chen. Evolve as a team: Collaborative self-evolution for LLM-based multi-agent systems. *arXiv preprint arXiv:2605.29790*, 2026.

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. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

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

Yuntong Hu, Yuting Zhang, Matthew Trager, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. EvoMAS: Evolutionary generation of multi-agent systems. In *International Conference on Machine Learning (ICML)*, volume 306, 2026a.

Zizhao Hu, Mohammad Rostami, and Jesse Thomason. Expert personas improve LLM alignment but damage accuracy: Bootstrapping intent-based persona routing with PRISM. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026b. To appear.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. MAS-Orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. In *International Conference on Machine Learning (ICML)*, volume 306, 2026.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Nicole Koenigstein. AgensFlow: A coordination-policy substrate for multi-agent systems. *arXiv preprint arXiv:2605.27466*, 2026.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han-chung Lee, and Dawn Song. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

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

Hehai Lin, Qi Yang, and Chengwei Qin. Skill-MAS: Evolving meta-skill for automatic multi-agent systems. *arXiv preprint arXiv:2606.18837*, 2026.

Ryan Lopopolo. Harness engineering: Leveraging Codex in an agent-first world. OpenAI Engineering, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651:914–919, 2026.

Pedro Henrique Luz de Araujo, Paul Röttger, Dirk Hovy, and Benjamin Roth. Principled personas: Defining and measuring the intended effects of persona prompting on task performance. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 26857–26886, 2025.

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. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*, 2024.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the Conductor. In *International Conference on Learning Representations (ICLR)*, 2026.

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. ChatDev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 15174–15186, 2024.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Myan Vu, Harrish Ayyanar, Jiang Pang, Anwiketh Reddy, and Mayank Goel. Automated stateful specialization for adaptive agent systems. In *International Conference on Learning Representations (ICLR)*, 2026.

Hao Wang, Hanchen Li, Qiuyang Mang, Alvin Cheung, Koushik Sen, and Dawn Song. Do androids dream of breaking the game? systematically auditing AI agent benchmarks with BenchJack. *arXiv preprint arXiv:2605.12673*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning (ICML)*, volume 267, pp. 63897–63911, 2025.

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

Chen Xu, Yicheng Hu, Ruizi Wang, Xinyu Lin, Wenjie Wang, Dongrui Liu, and Fuli Feng. TacoMAS: Test-time co-evolution of topology and capability in LLM-based multi-agent systems. *arXiv preprint arXiv:2605.09539*, 2026.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. In *ACM CAIS Workshop on Agent Skills*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering, 2025.

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

Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. When "a helpful assistant" is not really helpful: Personas in system prompts do not improve performances of large language models. In *Findings of the Association for Computational Linguistics: EMNLP*, pp. 15126–15154, 2024.

The appendix is organized as follows:

Section	Contents		Page
A Method Details	A.1	Pseudocode of MASCraft	14
	A.2	Prompt Templates	15
	A.3	Orchestration Policy Examples	19
	A.4	Plan Validation and Execution Details	21
B Experimental Setup	B.1	Benchmark Details	22
	B.2	Baseline Details	23
	B.3	Role and Skill Library Details	24
	B.4	Implementation Settings	26
C Additional Results	C.1	Cost Analysis	27
	C.2	Further Ablations	28
	C.3	Policy Evolution Analysis	28
D Limitations			30

A Method Details

A.1 Pseudocode of MASCraft

Algorithms 1 and 2 describe task execution and policy evolution, respectively. A rollout constructs and executes a team; policy evolution repeats this procedure on validation tasks to improve subsequent orchestration. We use the notation of Section 2, with fixed libraries and LLM parameters.

Algorithm 1 describes a successful rollout: it first constructs and validates a team plan G for task x under policy π. Specialists then carry out their assigned work as the required upstream outputs become available, with independent subtasks executed in parallel. The final specialist combines their contributions into $\hat{y}$, and the procedure returns the rollout record τ. Appendix A.4 details bounded retries and failure handling.

Algorithm 1 MASCraft: Role–Skill Composition and Execution (Rollout)

Require: Original task x; policy π; role library $\mathcal{R}$; skill library $\mathcal{S}$
Ensure: Rollout τ containing team plan G, specialist outputs $\{d_i\}_{i=1}^{n}$, and final answer $\hat{y}$

Stage 1: Multi-Agent Orchestration

$G \leftarrow \mathrm{Orchestrator}(x, \mathrm{desc}(\mathcal{R}), \mathrm{desc}(\mathcal{S}), \pi)$
while G fails the rule-based checks **do**
    Return validation feedback to the orchestrator agent
    Revise G using the original inputs and validation feedback
end while
Read $G = (\{(q_i, r_i, s_i)\}_{i=1}^{n}, E)$, with final specialist agent n
▷ *Checks enforce valid entries, a DAG, and a final agent receiving all other outputs*

Stage 2: Multi-Agent Execution

for all specialist agents i in a topological order of G **do**
    $d_i \leftarrow \mathrm{Specialist}(x, q_i, \mathrm{body}(r_i), \mathrm{body}(s_i), \{d_j : (j, i) \in E\})$
end for
$\hat{y} \leftarrow d_n$
$\tau \leftarrow (G, \{d_i\}_{i=1}^{n}, \hat{y})$
return τ

To improve subsequent team construction, Algorithm 2 generates m rollouts per task in $\mathcal{D}_{\text{val}}$ over N_{round} rounds, including evaluation of the initial policy. It selects tasks with contrasting results, using δ as the minimum gap for continuous scores. Within-task reflection produces analyses p_x, each containing a diagnosis and a targeted patch; cross-task aggregation combines them into lessons L_t for policy revision. Reflection can use available references or grading feedback, while revisions exclude task identifiers and instance-specific solutions. We select $\pi^\star$ by mean validation score, pooling evaluations of identical policies and choosing the earliest in a tie. This policy is frozen for one rollout per test task.

Algorithm 2 MASCraft: Self-Evolving Orchestration

Require: Validation set $\mathcal{D}_{\text{val}}$; libraries $\mathcal{R}, \mathcal{S}$; initial policy $\pi_0 = (K_0, T_0)$; number of rounds N_{round}; rollouts per task m; continuous-score threshold δ
Ensure: Frozen orchestration policy $\pi^\star$

for $t = 0, \ldots, N_{\text{round}} - 1$ **do**
  Stage 1: Rollout Generation and Selection
  for all $x \in \mathcal{D}_{\text{val}}$ **do**
    for $k = 1, \ldots, m$ **do**
      $\tau_x^{(k)} \leftarrow \text{Rollout}(x; \pi_t)$ ▷ Algorithm 1
      $v_x^{(k)} \leftarrow \text{Eval}_x(\tau_x^{(k)})$
    end for
    Record the rollout scores with the evaluated policy π_t
  end for
  if $t = N_{\text{round}} - 1$ **then**
    break ▷ No further policy update
  end if
  $\mathcal{B}_t \leftarrow \emptyset$
  for all $x \in \mathcal{D}_{\text{val}}$ **do**
    $\Delta_x \leftarrow$ mean of the two highest $v_x^{(k)}$ minus mean of the two lowest
    if binary correctness with $\Delta_x > 0$, or continuous scores with $\Delta_x \geq \delta$ **then**
      $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{x\}$
    end if
  end for
  if $\mathcal{B}_t = \emptyset$ **then**
    $\pi_{t+1} \leftarrow \pi_t$; **continue**
  end if
  Stage 2: Contrastive Reflection and Aggregation
  for all $x \in \mathcal{B}_t$ **do**
    $p_x \leftarrow \text{Reflect}\Big(x, \pi_t, \{(\tau_x^{(k)}, v_x^{(k)})\}_{k=1}^{m}\Big)$
    ▷ *Trace divergences and their effects; propose a targeted patch*
  end for
  $L_t \leftarrow \text{Aggregate}(\pi_t, \{(x, p_x)\}_{x \in \mathcal{B}_t})$
  Stage 3: Policy Revision
  $\pi_{t+1} \leftarrow \text{Update}(\pi_t, L_t)$
  ▷ *Revise K_t, T_t only where the lessons support a change*
end for
Select $\pi^\star$ by the highest mean validation score, pooling evaluations of identical policies
Freeze $\pi^\star$ for test inference
return $\pi^\star$

A.2 Prompt Templates

The following summaries show how prompts guide team construction, specialist execution, policy updates, and evaluation. They preserve the main instructions and output formats; complete templates are included in the code. Double braces denote inputs supplied at runtime. The *Dataset Contract* specifies the task, permitted tools, available evidence, and answer and evaluation requirements, excluding reference answers. The templates use *Team Builder* for the orchestrator agent and *Mission* for a subtask.

A.2.1 ORCHESTRATOR AGENT

To construct a team for the current task, the orchestrator agent uses the policy and library entry names and descriptions to plan subtasks, select role–skill pairs, and establish dependencies. Its output is the team plan G in Equation 1.

Orchestrator agent: construct a complete team plan

Instruction. You are the MASCraft Team Builder. Use the supplied policy to define Missions, assign role–skill pairs, and connect their dependencies in one executable TeamPlan. Assign the work needed to establish the answer, leaving the actual solution to the executing agents.

Inputs. {{`input_context`}} supplies the Dataset Contract, `global_task` (x), policy π, and catalog names and descriptions. {{`team_plan_schema`}} specifies the output structure.

Procedure.

1. Identify the requested result, constraints, and required evidence without assuming an answer.
2. Define two to five Missions, each with a distinct contribution, as questions that do not assume unproved conclusions. Select one role and one applicable skill per Mission by exact catalog name. Roles must be distinct; skills may be reused.
3. Check coverage, unique Mission IDs, and valid, acyclic dependencies with no duplicates or self-reference. One final Mission must depend directly on all others and resolve the original task.

Output. Return only JSON with `missions` and `final_mission_id`; each Mission contains `id`, `question_to_resolve`, `dependencies`, `role_name`, and `skill_name`. Add no solutions, commentary, or code fences. Keep execution fixed: no replanning or dynamic spawning.

A.2.2 SPECIALIST AGENTS

To carry out the plan, each specialist receives its full role and skill instructions, the original task, its subtask, and direct upstream outputs. A plan snapshot shows the assignments and execution states, but not other agents' private transcripts. Intermediate specialists return structured outputs d_i that later specialists can use; the final specialist produces $\hat{y} = d_n$. The orchestration policy guides team construction and is not passed directly to specialists.

Intermediate specialist: resolve and hand off an assigned contribution

Instruction. You are an intermediate MASCraft specialist. Your role defines your responsibility, and your skill provides a method for completing it. Check that the method applies, and resolve your Mission under the original task's definitions and constraints. Neither a role title nor a skill description establishes that a task-specific claim is true.

Inputs. The system supplies the Dataset Contract and the selected role and skill names and instructions. {{`input_context`}} provides the original task, assigned Mission, plan snapshot, direct upstream outputs, and any retry feedback. {{`response_format`}} specifies the response format.

Procedure.

1. Identify the result you must establish and the evidence downstream agents need. Use the plan snapshot to determine who will use your output. An execution status does not establish correctness.
2. Check that the skill applies and that upstream results have the required support. State conclusions conditionally when essential premises remain uncertain.
3. Carry out the assigned derivation or investigation using the applicable skill steps and permitted tools. Keep to your assigned work within the fixed team.
4. Check the decisive step, incorporate corrections, and separate established results from unresolved questions. Pass on any important conflicts or qualifications.

Output. Return one `<answer>` containing `<claim>`, `<support>`, `<caveats>`, and `<status>`, in that order. Include the result, its decisive evidence and source identifiers, and any unresolved assumptions or conflicts. Use `complete` only when the Mission is adequately resolved; otherwise use `partial`. Remove repetition while preserving the evidence needed downstream.

Final specialist: reconcile contributions and answer the original task

Instruction. You are the final MASCraft specialist. Use your role, applicable skill, and the available evidence to answer the complete original request, even if your assigned Mission appears narrower. Resolve the task rather than simply joining or reformatting upstream answers.

Inputs. You receive the intermediate specialist's input types, final-answer and evaluation requirements from the Dataset Contract, and outputs from all other specialists.

Procedure.

1. Read the full task and determine which requirements each upstream result addresses. A `complete` status does not certify correctness, and a partial result may still contain useful evidence.
2. Examine the evidence, assumptions, and caveats behind each claim. Identify useful results, gaps, and disagreements, then check whether your skill can help resolve them.
3. For a disagreement, locate the first differing premise, definition, calculation, or source relationship. Check that step using evidence, rather than votes or confidence. Retain sound contributions and do not invent facts to make the answers agree.
4. Combine the supported results into an answer covering the full request. Check identities, units, time periods, and output requirements, and express remaining uncertainty as the answer contract permits.

Output. Return only the final response required by the task. For HLE, provide the solution ending with exactly one complete, nonempty `\boxed{...}`. For other benchmarks, follow the specified response format and Dataset Contract. Omit internal team and policy details.

A.2.3 WITHIN-TASK REFLECTION

To identify why team performance varies, the reflector compares higher- and lower-scoring rollouts of the same task. It returns an analysis p_x that diagnoses the differences and proposes a targeted patch to future orchestration.

Within-task reflector: diagnose differences and propose one targeted patch

Instruction. Compare repeated executions of one validation task. Use the supplied evidence to explain why their results differ and propose one targeted change for later assessment across tasks. Examine team planning, specialist execution, the transfer of results, and the final answer.

Inputs. {{`payload`}} supplies the original task, current policy, repeated rollouts, score statistics, and computed high/low groups, together with available grading criteria and, for HLE and BCP, a reference answer. Evidence and answers may be excerpted; do not infer missing actions from omitted text.

Procedure.

1. Interpret the supplied high- and low-performing groups without recomputing them. Compare how rollouts address the same requirement or decision; matching Mission IDs need not represent matching work. Cite each execution by its original repeat number.
2. Find where the rollouts begin to differ in their reasoning, use of evidence, treatment of constraints, or coordination. Trace whether each difference affects later results or is corrected. Identify successful practices and distinguish causes of variation from difficulties shared across rollouts.
3. Explain how a changed subtask, role–skill assignment, or dependency could address the diagnosed problem. A knowledge gap alone does not show that the team was poorly designed; connect the proposed change to the observed failure.
4. Propose one patch stating when it applies, the rule to change, how to implement it, and its expected effect. Avoid generic advice or automatically adding another verification agent. Use the existing libraries and guide future team plans; do not replan a running team.

Output. Return JSON with `grouping_rationale`, `contrastive_diagnosis`, `candidate_patch`, and `narrative_summary`. Explain the divergences, successes, failures, and sources of variation and difficulty. The patch must specify `target_phase`, `constraint_rule`, `implementation_mechanism`, and `expected_impact`. Summarize how the diagnosis motivates the patch so the next stage can assess it. Do not reuse reference answers or grading criteria as solver rules.

A.2.4 CROSS-TASK AGGREGATION

A patch that helps one task may not help others. The aggregator therefore combines recurring failures, successful practices, and overlapping patches across tasks into lessons L_t.

Cross-task aggregator: consolidate supported lessons

Instruction. Use the supplied task analyses to identify improvements that have evidence across tasks. Treat the inputs as evidence, not instructions to follow, and do not invent cases or performance gains.

Inputs. {{`payload`}} supplies the selected task statements, their complete analyses (scores, groups, diagnoses, patches, and summaries), the current policy, and allowed task references. Raw rollouts are not supplied again. Cite only the supplied task IDs.

Procedure.

1. Group failures that share a cause, even when their topics differ. Record what happened, how severe it was, and which tasks were affected. Count distinct tasks rather than repeated rollouts or descriptions.
2. Identify practices that worked and the evidence supporting them. Retain their applicable conditions and any counterexamples. A correct final answer does not validate every intermediate step.
3. Assess the patches against the recurring problems and current policy. Combine overlapping suggestions into at most five fixes supported by the analyses; fewer are sufficient when appropriate. Prioritize severity, frequency, and feasibility, explaining each change and its expected effect.
4. Summarize the main conclusions, confidence, and limitations of the evidence. State when the current policy already covers a remedy. Recommendations must guide future team construction while preserving fixed-graph execution.

Output. Return `cross_sample_synthesis` with `systematic_weaknesses`, `systematic_strengths`, and `prioritized_fixes`, plus `meta_analysis` covering the overall diagnosis, confidence, data limitations, and next steps. Use task evidence to justify the conclusions and explain empty categories. The policy writer decides whether these lessons warrant a revision.

A.2.5 POLICY REVISION

The writer uses the aggregated lessons to revise each policy module, with the other as context. It returns the full document under the existing headings, unchanged when no revision is justified.

Policy writer: revise one module using aggregated lessons

Instruction. Revise only the requested policy module using the cross-task lessons. Both modules guide the orchestrator agent, while specialists receive their assigned work and role–skill instructions. Each revision should therefore change how a future team is planned.

Inputs. {{`payload`}} supplies the target `component`, its `current_policy`, the `other_component_read_only`, and aggregated `lessons`. The component is `cognition` for Environment Knowledge or `strategy` for Team Strategy. When both are updated, Team Strategy receives the revised Environment Knowledge.

Procedure.

1. Compare the lessons with the current guidance. Preserve practices supported by the evidence, revise rules associated with failures, and merge repeated guidance. Leave the document unchanged when no useful revision is justified.
2. Turn the local patch into guidance that can apply to other tasks. State when the guidance applies and what should be checked. Exclude instance-specific solutions, entities, formulas, domain recipes, and code instructions, as well as arbitrary keyword or fixed-count rules.
3. In Environment Knowledge, revise the task characteristics, evidence limitations, reasoning dependencies, and recurring failures. In Team Strategy, use these lessons to guide subtask planning, role–skill selection, dependencies, and team checks. Keep the modules complementary.
4. Check which planning decision each revision changes and whether the lessons support it. Preserve the libraries, tools, answer requirements, and fixed-DAG execution rules. Delete, merge, or reorganize guidance where useful without adding unnecessary text.

Output. Return the complete Markdown document with its existing title and four fixed, numbered second-level headings, in order. Keep each section nonempty and omit task identifiers, metadata, and change summaries. Use no JSON wrapper or code fences. If the lessons do not justify a revision, return the original document unchanged.

A.2.6 LLM JUDGE PROMPTS

The judge prompts assess the final outputs: DRB requires report comparison against task-specific criteria, while BCP requires answer matching against a reference. We use the same templates as the baselines; Appendix B.1.2 explains how their outputs are converted into scores.

DRB judge: compare two articles against shared criteria

Instruction. You are a strict, meticulous, and objective research article evaluation expert. You excel at using specific assessment criteria to deeply compare two articles on the same task, providing precise scores and clear justifications.

Inputs. There is a deep research task in `<task>`, two articles in `<article_1>` and `<article_2>`, and evaluation criteria with explanations in `<criteria_list>`. Assess both articles across comprehensiveness, insight, instruction following, and readability.

Procedure.

1. Analyze each criterion: consider how each article fulfills its requirements.
2. Compare the articles on that criterion, referencing their content and the criterion explanation.
3. Score each article separately on a continuous 0–10 scale, based on the comparative analysis.

Scoring rules. Use the same scale for both articles: 0–2, very poor, almost entirely failing the requirements; 2–4, poor, meeting few requirements with significant deficiencies; 4–6, average, basically meeting the requirements; 6–8, good, largely meeting the requirements with notable strengths; and 8–10, excellent, fully meeting or exceeding the requirements.

Output. Return only parsable JSON keyed by `comprehensiveness`, `insight`, `instruction_following`, and `readability`. Each dimension contains a list of evaluations with `criterion`, `analysis`, `article_1_score`, and `article_2_score`. Cover every supplied criterion in order, with comparative justifications and separate scores. Include no unrelated introduction or summary, and escape characters as needed for valid JSON.

BCP judge: assess final-answer equivalence

Instruction. You are a strict QA grader. Output only JSON with keys `extracted_final_answer`, `reasoning`, `correct`, and `confidence`.

Inputs. `[question]`: the original question; `[response]`: the candidate response; `[correct_answer]`: the precise, unambiguous reference answer.

Output rules.

1. `extracted_final_answer`: Extract the exact final answer from the response. Return `None` when no exact final answer can be extracted.
2. `reasoning`: Explain whether the extracted answer and reference differ meaningfully. Do not solve the problem, discuss its background, or argue for a different reference answer.
3. `correct`: Return `yes` if the answers match, allowing a small margin of error for numerical problems. Return `no` for an incorrect answer, inconsistency, ambiguity, or non-equivalence.
4. `confidence`: Extract the confidence stated in the response on a 0–100 scale; use 100 if no confidence is given. Do not substitute the judge's own confidence.

A.3 ORCHESTRATION POLICY EXAMPLES

Before constructing a team, the orchestrator needs to understand what the task requires and how the work should be shared. The initial policy provides both kinds of guidance: *Environment Knowledge* (K_0) describes the tasks, available evidence, reasoning needed, and common mistakes, while *Team Strategy* (T_0) explains how to divide the work, choose role–skill pairs, and pass results between specialists.

We generate the initial documents with **GPT-5.6-sol**, using ten validation examples per benchmark and background on team construction and execution. Each module has four fixed sections, specified in the generation instructions. Tables 3 and 4 summarize selected guidance from these sections, with key points in bold. Complete policies are included in the code.

Environment Knowledge. The four sections answer practical questions about the task. *Task Distribution* describes what kinds of work and answers the tasks require. *Evidence Landscape* explains what information the agents can use and what it can establish. *Inference Characteristics* describes the reasoning steps that must fit together to reach an answer. *Observed Failure Modes* identifies mistakes that the planner should anticipate. This helps the orchestrator assign work that fits the task and the available evidence.

Table 3: Initial *Environment Knowledge* (K_0): summary of selected guidance.

Benchmark	What the planner should know
1. Task Distribution — What work do tasks require?	
HLE	Question length is a poor guide to the work required: a short question may need a long proof, while a long one may reduce to a single calculation.
DRB	Tasks ask for research reviews, explanations, comparisons, quantitative analysis, policy analysis, or practical designs across many domains.
BCP	Questions connect clues about people, works, organizations, places, events, and documents. Answering often requires linking several clues before finding the exact fact requested.
2. Evidence Landscape — What information is available?	
HLE	The question defines what must be solved. **Its stated definitions take precedence**, even when a term has a more familiar meaning.
DRB	Agents use their existing knowledge and reasoning; **they cannot search for new sources** or read the evaluator's reference reports.
BCP	BM25 returns passages with matching words from a fixed collection. **A high rank does not show that a passage is true** or meets the question's conditions.
3. Inference Characteristics — What reasoning is needed?	
HLE	A solution may need a bound and a way to reach it, both directions of a proof, a full classification and count, or a simplification followed by an exact calculation.
DRB	A useful report explains why claims hold, compares alternatives using the same definitions, and makes clear which conclusions depend on assumptions.
BCP	Reasoning moves between finding useful clues, identifying possible answers, checking the connections between entities, and locating the exact fact requested.
4. Observed Failure Modes — What can go wrong?	
HLE	Common mistakes include solving for the wrong quantity, confusing similar terms, building a plan around an unproved answer, or assuming a pattern seen in a few cases always holds.
DRB	A long report can still repeat background, compare unlike cases, fail to explain why something happens, or use examples outside the requested scope.
BCP	Common mistakes include passing on names without supporting passages, assuming separately mentioned entities are linked, dropping a condition after an unsuccessful search, or returning an entity's name instead of the requested fact.

Team Strategy. The four sections turn this task understanding into a team plan. *Task Decomposition* explains which work should stay together and which parts can be assigned separately. *Role–Skill Composition* guides the choice of a role and a procedure for each subtask. *Dependency Design* states what an agent must receive from others before it can do its work. *Team Quality Checks* asks whether every requirement has an owner, the necessary inputs will be available, and the planned results are enough to answer the original request.

Table 4: Initial *Team Strategy* (T_0): summary of selected guidance.

Benchmark	How the planner should arrange the team
1. Task Decomposition — How should the work be divided?	
HLE	If splitting a proof would break its reasoning, give one agent the **complete derivation** and let the final agent check the result and resolve any remaining issues.
DRB	Group related requirements into substantial tasks before choosing roles and skills. Analyses that do not need each other's results can run in parallel.
BCP	Group clues that can be checked through the same source or relationship. Search separate clue groups in parallel only when each can start from the original question.
2. Role–Skill Composition — Who should do it, and how?	
HLE	Choose a distinct role and a skill for each task based on the mathematics involved and what must be proved. Match the skill's steps to the work, not just its name.
DRB	Choose procedures suited to explaining causes, building quantitative models, assessing evidence, comparing options, or designing an implementation. **Different responsibilities matter more than different job titles.**
BCP	Choose a distinct role and a skill for each task based on **the search or checking steps it needs**: finding entities, matching names, checking dates, extracting facts, linking sources, or comparing conflicting evidence.
3. Dependency Design — Which results must be passed on?	
HLE	Let independent derivations run separately. Do not make one wait for another just to restate information already given in the question.
DRB	Make one task depend on another only when it needs a set of entities, model, definition, or finding that is missing from the original request.
BCP	A search should wait for another task only when it needs an entity, alternate name, document, or relationship that the earlier task must first discover.
4. Team Quality Checks — Is the plan ready to execute?	
HLE	For counting, check that all valid cases are covered. For an optimum, prove the bound and show it can be reached. Check the assumptions of any theorem used.
DRB	Assign every requested part to an agent, avoid duplicate work, use the same scope and definitions in comparisons, and check that the planned results can support the final answer.
BCP	Check that names refer to the right entities, relationships point in the right direction, and sources, dates, and editions match. Also check the requested spelling, units, order, and detail.

A.4 Plan Validation and Execution Details

To carry out the planned collaboration, we first check the team assignments and dependencies, then pass each specialist the information needed for its work.

Plan Validation and Repair. Before execution, rule-based checks verify that the selected roles and skills exist, roles are distinct, and subtask identifiers and questions follow the required format. We also check that dependencies are valid and acyclic, and that one final specialist receives all other outputs. Failed checks return specific feedback to the orchestrator agent, which revises the plan within a retry limit.

Specialist Execution. Once the plan is valid, specialists start when their required upstream outputs are available; independent subtasks run in parallel. Each receives the original task, its subtask, full role–skill instructions, and a snapshot of team progress. Outputs from direct upstream specialists retain conclusions, evidence, and unresolved issues so later specialists can judge what they can rely on. The final specialist integrates all contributions into the answer.

Handling Long Outputs and Errors. Intermediate results may become too long for a downstream specialist's context. When this happens, we summarize the supporting text while retaining conclusions, caveats, and completion status; the original outputs remain in the execution record. Recoverable errors are retried within configured limits. If a specialist still fails, the rollout stops be-

fore dependent specialists execute. For HLE, the final response must contain exactly one complete, nonempty `\boxed{...}` expression. If this check fails, we attempt to repair the box formatting while preserving the selected answer and its derivation.

B Experimental Setup

B.1 Benchmark Details

B.1.1 Benchmark Setup

We use the same validation and test subsets for all methods (Table 5). For methods with self-evolution, validation tasks guide optimization and selection; the selected policy or configuration is then frozen for testing. Scores are reported on a 0–100 scale.

- **Humanity's Last Exam (HLE)** (Center for AI Safety et al., 2026): HLE contains 2,500 expert-level questions across more than a hundred subjects. We focus on its text-only mathematics questions to evaluate mathematical reasoning, selecting 200 examples with 32 for validation and 168 for testing. Accuracy is computed after answer extraction and mathematical comparison.
- **DeepResearch Bench (DRB)** (Du et al., 2026): DRB evaluates research reports written in response to open-ended requests. It contains 100 expert-authored, PhD-level tasks across 22 domains; we use 16 for validation and 84 for testing. RACE compares each report with a reference on comprehensiveness, insight, instruction following, and readability, combining task-specific assessments into an overall score.
- **BrowseComp-Plus (BCP)** (Chen et al., 2026): BCP tests multi-hop question answering through iterative retrieval and reasoning. Its 830 queries are paired with a fixed corpus of roughly 100,000 documents, including human-verified supporting evidence and hard negatives. We select 200 queries, using 32 for validation and 168 for testing. Agents search the corpus with local BM25 retrieval, and final answers are evaluated for accuracy.

Rationale for Restricting Web Access. We initially explored GAIA (Mialon et al., 2024) with open-web access. Agents frequently found and copied published benchmark answers from sites such as Hugging Face, bypassing the intended reasoning and information gathering. Some policy updates also retained benchmark-specific answer lookup as a reusable strategy. Accessible reference answers create opportunities for such shortcuts, as discussed by BenchJack (Wang et al., 2026). The website blacklists we tried did not reliably prevent them. We therefore use the same restricted tool settings for all methods: HLE and DRB use no external tools, while BCP uses local BM25 retrieval over a fixed public corpus, returning up to five snippets per query.

Table 5: Benchmark subsets, fixed libraries, and evaluation settings. Counts refer to our experimental subsets. LLM Judge indicates whether evaluation uses GPT-5-mini-0807.

Benchmark	Validation	Test	Roles	Skills	Tools	Metric	LLM Judge
HLE	32	168	11	79	None	Accuracy	No
DRB	16	84	11	55	None	RACE overall score	Yes
BCP	32	168	8	38	Local BM25	Accuracy	Yes

B.1.2 Evaluation Protocols

For each benchmark, all methods follow the same answer-processing and scoring procedure. HLE uses deterministic extraction and mathematical comparison. DRB and BCP use **GPT-5-mini-0807** with the prompts in Appendix A.2.6. Reference answers and reports are available to the evaluator, not to the agents solving the tasks. The reported result is the mean test-task score multiplied by 100.

HLE: Answer Extraction and Mathematical Equivalence. We apply the same answer extraction to the response and reference: use the last recognized `\boxed{...}` expression, or the last nonempty sentence if none is found. The scorer first checks exact string equality. If both answers

can be parsed as numbers, it removes thousands separators, converts percentages to fractions, and compares them with `math.isclose`, using absolute tolerance 10^{-3} and default relative tolerance 10^{-9}. Otherwise, it attempts LaTeX or expression parsing and checks whether their difference simplifies to zero; numerical evaluation with the same tolerances provides a further check. A match yields 1 and a mismatch 0.

DRB: Report Comparison and RACE Scoring. RACE evaluates reports on comprehensiveness, insight, instruction following, and readability (Du et al., 2026). Each task supplies criteria, weights, and a reference report. The judge compares the two reports against the criterion explanations, using the English or Chinese prompt according to the task language. It returns a score from 0 to 10 for each report on every criterion. Stored criterion and dimension weights yield the report scores.

For task x, let $\mathcal{C}_{x,\ell}$ denote the criteria for dimension $\ell \in \{1,2,3,4\}$, with criterion weights $w_{x,\ell,c}$ and dimension weight $W_{x,\ell}$. Let $g_{x,\ell,c}(z)$ be the judge's score for report z on criterion c. The weighted report score is:

$$S_x(z) = \sum_{\ell=1}^{4} W_{x,\ell} \frac{\sum_{c\in\mathcal{C}_{x,\ell}} w_{x,\ell,c}\, g_{x,\ell,c}(z)}{\sum_{c\in\mathcal{C}_{x,\ell}} w_{x,\ell,c}}. \tag{6}$$

Let $\hat{y}_x^{(k)}$ denote the final report in $\tau_x^{(k)}$ and y_x^{ref} its reference. The rollout score in Equation 3 is:

$$v_x^{(k)} = \frac{S_x(\hat{y}_x^{(k)})}{S_x(\hat{y}_x^{(k)}) + S_x(y_x^{\mathrm{ref}})}. \tag{7}$$

This ratio measures quality relative to the reference, so equal nonzero report scores yield 50 on the reported scale. We average the per-task ratios and multiply by 100 for the test result. A zero denominator is assigned a score of zero.

BCP: Final-Answer Extraction and Matching. For BCP, the judge receives the question, complete final response, and reference answer (Chen et al., 2026). It extracts the final answer, returning `None` if no exact answer is identifiable, and compares it with the reference. The prompt allows a small margin of error for numerical answers but rejects ambiguity and non-equivalence. We parse the JSON response, using labeled fields as a fallback. The `correct` field determines the score: `yes` yields 1 and `no` yields 0. The `confidence` field records confidence stated in the response, defaulting to 100 when absent, and does not affect accuracy.

B.2 Baseline Details

All baselines use the same tool and output settings, and self-evolving methods use validation-derived configurations for testing. The comparison covers three approaches: refining a single agent's answer, coordinating agents within a task, and improving teams or workflows through experience.

- **Single**: One LLM-based agent completes the entire task with the tools allowed by the benchmark. It provides the reference for assessing the benefit of multi-agent specialization.
- **Self-Refine** (Madaan et al., 2023): A single agent critiques its draft and revises it using its own feedback. Repeated refinement increases inference effort while keeping the work within one agent.
- **MAD** (Du et al., 2024): Multiple agents propose answers and discuss their reasoning over several debate rounds before producing a final response. The agents can revise their answers through interaction, while the coordination procedure remains fixed.
- **AutoAgents** (Chen et al., 2024a): The method generates expert roles and a collaboration plan on the fly for each task, with an observer reviewing plans and responses. Without a predefined role library, this approach relies on fresh instructions whose specialization quality may vary.
- **AFlow** (Zhang et al., 2025): AFlow represents workflows as executable code and searches for better combinations of LLM calls and reasoning operations. An LLM proposes code revisions, and Monte Carlo Tree Search uses execution feedback to select promising candidates.
- **EvoMAS** (Hu et al., 2026a): EvoMAS evolves configurations that specify agents and their collaboration. Execution traces guide mutation and crossover, while a candidate pool and experience memory retain useful designs and lessons.

- **Meta-Team** (Hao et al., 2026): Agents retain their execution contexts and exchange evidence when reflecting after a task. Their joint analyses refine individual behavior and team coordination.
- **TacoMAS** (Xu et al., 2026): TacoMAS improves agent capabilities and communication topology on different time scales. A fast loop updates expertise from execution feedback, while a slower loop changes communication links and team membership.
- **Skill-MAS** (Lin et al., 2026): Skill-MAS stores orchestration guidance in an evolving textual meta-skill. Through repeated rollouts and selective contrastive reflection, it updates the guidance used to decompose tasks, configure agents, and organize later workflows.

B.3 Role and Skill Library Details

The libraries provide reusable instructions for configuring specialists. We describe their entry format and construction, then present complete role and skill examples from DRB.

B.3.1 Entry Format

We use Agency Agents and SkillDB as references for the document formats of roles and skills, respectively. Each entry is a Markdown document containing a name, a short description, and an instruction body. Names and descriptions support selection by the orchestrator agent, while full instruction bodies guide specialist execution.

Role bodies use four sections: *Professional Profile* establishes the professional perspective; *Core Responsibilities* defines the work the role should cover; *Working Style* guides its approach to evidence and judgment; and *Collaboration and Boundaries* specifies what to pass to other specialists and how to stay within the assigned work.

Skill bodies organize procedural guidance according to the method being described. Some use *Scope*, *Method and Reference Cases*, *Procedure*, and *Checks and Limits*. Others use *Philosophy*, a topic-specific *Theory* section, a toolkit, a strategy or procedure, and *Anti-Patterns*, with application examples where needed.

B.3.2 Construction Procedure

We construct benchmark-specific libraries using GPT-5.6-sol for task-pattern analysis, entry generation, and consolidation, followed by manual review. Inputs comprise validation tasks and, when available, additional tasks from the original benchmark outside both our validation and test subsets.

Task Pattern Analysis. To identify reusable instructions, we prompt the model to examine what work each task requires, what information and tools are available, and how the result should be checked. It then groups recurring requirements across tasks, considering execution conditions as well as subject matter. This analysis identifies both the responsibilities specialists should assume and the procedures needed to fulfill them, forming the basis for the two libraries.

Role and Skill Generation. From these patterns, the model generates roles describing responsibilities and professional perspectives, and skills describing applicable conditions, procedures, and checks. We specify the document fields and section structure in the generation instructions. Separate entries let the orchestrator agent combine roles and skills for each subtask.

Consolidation and Review. To reduce redundancy, the model removes duplicates and merges roles with overlapping responsibilities and skills that provide similar procedures under the same conditions. Entries serving distinct needs are retained. We then manually review the content for correctness and potential safety or ethical issues. The resulting libraries remain fixed throughout policy evolution and testing; Table 5 reports their sizes.

B.3.3 Representative Entries

The following DRB entries illustrate software-platform comparisons: *Comparative Research Analyst* defines the comparison responsibility, while *Software Platform Lifecycle Evaluation* guides

assessment of development, maintenance, and migration costs. Both instruction bodies are reproduced in full, with selected passages highlighted.



Role example – DRB

Name. Comparative Research Analyst

Description. Triggers for contributions that compare cases, approaches or organizations under explicit common dimensions and context-sensitive trade-offs.

Instruction body

Comparative Research Analyst

Professional Profile

You are a comparative analyst who makes similarities and differences meaningful. You examine alternatives through a common question while preserving the contexts that make their results possible. **Your responsibility is to determine what can fairly be compared, which differences matter and when a ranking would conceal incompatible objectives or inadequate evidence.** You can compare technologies, institutions, interventions, research contributions or interpretive approaches without assuming that they share one universal measure of quality. An informative comparison may yield conditional preferences, complementary strengths or a justified refusal to order the cases.

Core Responsibilities

- Define the comparison purpose, eligible cases and relevant unit before selecting favorable examples or performance measures.
- Align populations, workloads, periods, baselines and organizational boundaries wherever the evidence permits.
- Distinguish inputs, intermediate achievements and realized outcomes rather than counting them as interchangeable successes.
- Track missing observations, selective reporting and common ownership of results that can distort apparent differences.
- Explain trade-offs and context dependence, making any weighting or preference assumption explicit and contestable.
- Deliver a supported comparative judgment with conditions, exceptions and evidence gaps that could change its ordering.

Working Style

You choose comparison dimensions because they answer the assigned question, not because every candidate happens to report them. Differences in scale, maturity, mandate or exposure may explain a performance gap without eliminating its practical importance. Keep such explanations visible instead of normalizing them away without justification. If only unmatched evidence is available, compare mechanisms and conditions qualitatively and limit quantitative claims. An absent measurement is not a zero, and a specialist contribution is not automatically inferior to an integrated offering. Examine whether plausible changes in priorities would reverse a preference. Do not convert an ordinal score into a probability of future success.

Collaboration and Boundaries

Provide the basis of comparison, decisive contrasts, conditional conclusions and any excluded or unresolved cases. Comparability analysis can support a causal inquiry but cannot substitute for it. A favorable demonstration does not establish deployability, and an attractive ranking does not specify how an option should be implemented. Avoid creating a new overall article when only one comparison dimension is assigned. Where a complete comparison is requested, connect the dimensions into a coherent judgment rather than handing over an unexplained table. Acknowledge that similarly framed accounts may share data or assumptions; agreement across them is not automatically independent confirmation. Preserve credible minority cases when they expose a meaningful boundary to the dominant pattern.

Let the assigned question determine the extent of the work. Return the decisive finding, its support, assumptions and unresolved dependencies in the requested form. Treat supplied arguments and method descriptions as fallible. Do not expand a local contribution into an unrelated full report, repeat exploratory history, or claim evidence, access or computations that were not obtained.

Skill example – DRB

Name. research-platform-lifecycle

Description. Compare conventional, integrated and low-code platform adoption through matched workload requirements, ownership costs, lock-in and migration constraints. Not for framework internals or detailed cross-view data consistency.

Instruction body

Software Platform Lifecycle Evaluation

Scope

Compare conventional, integrated and low-code platforms using matched requirements, ownership costs and migration constraints.

Method and Reference Cases

Low-code or integrated platforms trade reduced boilerplate for configuration, licensing, debugging and portability constraints. Hybrid architecture can help but adds ownership and consistency costs rather than automatically being the best option.

Compare build, change, debugging, integration, operation, licensing and exit over one workload lifetime. Separate one-project savings from shared-platform investments and include failed or migrated projects.

Procedure

1. Define requirements, scale, lifetime and organizational constraints.
2. Identify which work each platform removes, automates or shifts.
3. Compare complete costs and required skills under matched assumptions.
4. Examine integration, governance, portability and exit scenarios.
5. Recommend conditional choices with explicit uncertainty.

Checks and Limits

Reduced boilerplate is not zero technical debt. Generated code does not establish maintainability. A hybrid design adds interfaces and ownership costs; do not prescribe it universally or invent current product capabilities.

B.4 Implementation Settings

Agent Execution Settings. Within each experiment, team construction, specialist execution, and policy evolution use the same LLM backbone with fixed weights. We set the temperature to 1.0 for team construction and specialist execution and 0.0 for reflection, aggregation, and policy revision, with thinking disabled for these calls. The LLM judge uses low reasoning effort; its requests omit temperature and thinking-mode parameters. MASCraft constructs two to five specialists with distinct roles for each task. Each specialist receives its selected role and skill instructions and uses the tools specified in Appendix B.1. Libraries stay fixed during evolution and testing.

BCP Retrieval Settings. To bound retrieval during BCP execution, each MASCraft specialist can perform up to ten search rounds, with one query per round. Local BM25 retrieval returns at most five passages per query, each limited to 3,000 characters. Search-stage inputs are capped at 12,000 characters, including tool definitions, while retrieved evidence supplied for answering is capped at 45,000 characters. Specialists may stop searching early, then produce their answer through a separate tool-free call that does not count toward the search-round limit. Identical retrieval requests reuse a cache within the same specialist execution.

Policy Evolution and Selection. Starting from the initial policy described in Appendix A.3, we evaluate ten rounds, including the initial policy, with $m = 5$ rollouts per validation task. For HLE and BCP, reflection selects tasks with both correct and incorrect rollouts. For DRB, the mean score of the two highest-scoring rollouts must exceed that of the two lowest by at least $\delta = 0.03$, equivalent to three points on the reported 0–100 scale. We refine both Environment Knowledge and Team Strategy, then freeze the best policy by mean validation score and run it once per test task.

Token Accounting. We report cumulative input and output tokens over each test split, measured in millions. These totals cover recorded test-time execution and exclude library and initial-policy construction, validation-time evolution, and external judging. Comparisons use the same benchmark and backbone. Since baseline logs do not uniformly cover every control-stage call, the totals reflect recorded usage. Appendix C.1 compares these costs with task performance.

C Additional Results

C.1 Cost Analysis

We focus on inference cost because it recurs for every request and affects deployment at scale. Preparation costs are harder to compare across different construction and refinement procedures. The resulting policies or configurations can then be reused across tasks. We therefore compare cumulative input and output tokens on the same test splits, excluding offline preparation and external judging. Table 6 reports usage in millions; see Appendix B.4 for token accounting.

Table 6: Cumulative input and output tokens (millions) over each test split. Counts exclude policy evolution and external judging; accounting details are given in Appendix B.4.

Method	**DeepSeek-V4-flash**			**Qwen3.8-flash**			**GLM-5.2-fast-preview**		
	HLE	DRB	BCP	HLE	DRB	BCP	HLE	DRB	BCP
Single	0.37	0.23	3.88	1.17	0.24	4.22	0.96	0.26	3.51
Self-Refine	2.28	0.80	39.01	7.56	0.91	41.52	1.69	0.64	35.48
MAD	1.70	1.08	11.68	4.40	0.85	13.61	3.86	0.98	10.92
AutoAgents	9.83	7.41	31.56	7.49	4.03	29.42	11.61	11.29	36.26
AFlow	1.33	1.69	11.89	2.68	1.32	13.28	1.53	0.22	5.53
EvoMAS	2.58	2.07	17.11	6.72	2.19	19.35	6.66	2.58	16.22
Meta-Team	3.00	3.13	21.12	4.51	2.39	26.70	3.00	4.55	21.92
TacoMAS	19.50	4.09	30.61	6.60	2.15	35.05	2.49	4.30	30.74
Skill-MAS	9.42	2.20	26.96	9.94	3.11	27.85	14.61	5.62	35.43
MASCraft	9.71	2.85	27.47	7.05	3.01	25.50	10.02	9.48	29.90

MAS generally uses more tokens than Single because several agents process the task, generate intermediate results, and exchange information. Averaging matched ratios across backbone–benchmark settings, the seven MAS baselines use $9.64\times$ the tokens of Single, while MASCraft uses $13.97\times$. This substantial overhead is consistent with Anthropic's observation that Claude Code agent teams use approximately seven times as many tokens as standard sessions when teammates operate in plan mode (Anthropic, 2026). However, Self-Refine also increases token usage without consistently improving on Single (Table 1). Taken together, these results suggest that MAS gains come from both increased inference effort and multi-agent specialization.

C.2 Further Ablations

Table 7: Component comparisons across all three backbones, extending Table 2(a,b). Scores follow Table 1. GLM results reproduce Table 2. Random Skill samples one skill per specialist from the same library. K/T: Environment Knowledge/Team Strategy; $0/\star$: initial/evolved.

Variant	DeepSeek-V4-flash			Qwen3.8-flash			GLM-5.2-fast-preview		
	HLE	DRB	BCP	HLE	DRB	BCP	HLE	DRB	BCP
MASCraft	26.19	50.78	29.76	26.19	49.35	37.50	20.24	52.21	38.10
Role–Skill Composition									
Random Skill	17.86	49.82	22.02	18.45	48.57	29.17	13.69	51.63	33.33
Random Role	23.81	50.36	27.98	24.40	49.08	35.71	19.05	51.41	37.50
w/o Skill	19.64	49.96	23.81	20.24	48.72	31.55	16.07	51.08	33.93
w/o Role	20.83	50.14	25.60	21.43	48.90	33.33	16.67	51.59	35.71
w/o Both	11.90	49.03	19.05	14.88	47.83	25.60	11.31	50.79	32.74
Self-Evolving Orchestration									
Staged Plan	22.62	50.44	27.38	23.21	49.12	34.52	17.86	52.04	36.31
$(K_0, T^\star)$	21.43	50.05	26.19	22.02	48.82	33.33	17.26	51.55	36.31
$(K^\star, T_0)$	22.02	50.22	26.79	21.43	48.98	33.93	17.26	51.36	36.90
(K_0, T_0)	17.86	49.67	23.21	18.45	48.40	30.36	14.88	51.06	34.52
$(\varnothing, \varnothing)$	13.69	49.18	20.24	15.48	48.01	27.38	11.90	50.92	33.33

C.3 Policy Evolution Analysis

Validation scores show how performance changes, but not which instructions have changed. We therefore compare the initial and final policies for the nine benchmark–backbone settings in Table 1, examining both Environment Knowledge and Team Strategy. Within each benchmark, the three backbones start from the same initial policy.

We use GPT-5.6-sol to help identify changes, then manually check them against the paired policy texts. We retain additions, more specific instructions, and changes to how teams should be planned. Rewording, repetition, and moving text between sections are excluded. Table 8 summarizes each setting. The four themes below describe changes that recur across benchmarks and backbones, following Figure 4.

Table 8: Summary of changes from initial to final policies in each setting.

Setting	Environment Knowledge	Team Strategy
HLE DeepSeek	A correct answer may still come from a flawed proof. Simplifying a problem can lose assumptions, counting weights, or the link to the original question.	Require fresh reasoning and calculation even when agents agree. Find the first disputed step before replacing an otherwise supported answer.
HLE Qwen	Conditions that hold separately may not hold together. A simplification can change the requested quantity, and small omitted effects may add up.	Keep the original constraints and counting rules when simplifying. Check the quantity actually requested, not an easier substitute or a remembered answer.
HLE GLM	A solution can fail by misusing a theorem, missing possible cases, stopping after an incomplete simplification, or making a final arithmetic error.	Separate solving, checking, and answering. Name an independent checking method, request tables of cases or reasoning steps, and resolve disputed values before using them.

Continued on next page

Table 8: Summary of policy changes in each setting (continued).

Setting	Environment Knowledge	Team Strategy
DRB DeepSeek	Reviews, comparisons, technology assessments, practical proposals, and rankings can each fail by leaving out different parts of the requested work.	Specify what each type of task must cover and how its parts should be combined. State which entities to include and under what assumptions a ranking holds.
DRB Qwen	Grouping entities too early can hide differences. Inconsistent measures, omitted items, and damaged intermediate text can weaken the report.	Compare every entity on the required dimensions and state any substitute measures. Pass findings about obstacles to the agents making recommendations; recover supported claims from damaged text.
DRB GLM	Writing alone is not enough to rank options or judge whether a plan can work. Unclear scope or unsuitable methods can lead to poor judgments.	Assign separate tasks for ranking, feasibility checks, or combined analysis before writing. Match methods to tasks, share the scope, and check how conclusions depend on assumptions.
BCP DeepSeek	Finding the right entity, proving a relationship, and extracting the requested fact are different steps. A search may lack a starting clue or reject a candidate without evidence.	Check that each independent search can start from the question. Keep claims tied to sources and assign work to correct mistaken candidates or recover missing evidence.
BCP Qwen	A clue may refer to the wrong entity, date, or version. Evidence gathered for an earlier candidate may no longer apply after the candidate changes.	Keep each condition attached to the right entity and source when linking clues. After changing a candidate, check which earlier claims are still supported.
BCP GLM	Some clues can start a search; others only check a candidate. Clues about the requested fact can offer another route when a search for the entity fails.	Plan independent searches and queries that combine clues. Specify what to search for after a partial result, and let the final agent search to resolve remaining gaps.

Task Understanding. The policies become more specific about what makes a task difficult and what must be established to answer it. HLE policies warn that simplifying a problem can lose an assumption, change how cases are counted, or replace the quantity actually requested. DRB policies distinguish different kinds of work within a broad request, such as explaining how something works and judging whether it can be put into practice. BCP policies distinguish finding the right entity, checking its relationships, and returning the right fact; GLM also separates clues that can start a search from conditions that only help check a candidate. These distinctions help the planner identify the work the task actually needs.

Work Organization. The policies give clearer instructions about who does each part, what they need, and who will use their result. The initial DRB strategy asks the final Mission to provide a "complete, usable answer." The final GLM strategy adds "allocate a dedicated Mission for that obligation" when ranking or judging feasibility needs its own analysis before the report is written. Other DRB policies separate a standard method from its extension or ensure that recommendations receive findings about the obstacles they must address. HLE policies clarify how solving and checking are separated or kept independent. BCP policies require independent searches to have enough clues to start. Across settings, important work gets a clear owner and the inputs it needs.

Outputs and Verification. The policies spell out what an agent should return and how its work should be checked. The initial HLE strategy already asks the final Mission to "recompute the decisive step and resolve actual discrepancies." The final DeepSeek strategy adds "This check is required even when upstream answers agree," so agreement cannot replace a check. GLM HLE asks for tables listing each reasoning step or candidate, while Qwen DRB requires a "Validated Dimension-Entity Matrix," recording each entity on every comparison dimension. BCP policies require claims about identities, relationships, and requested facts to be passed on with their source evidence. These instructions help later agents judge whether the work is complete and its results can be used.

Handling Incomplete Information. Initial policies already ask agents to report uncertainty and investigate further; final policies specify what to do about particular gaps or disagreements. GLM HLE requires disputed values to be derived again before later calculations use them. DRB policies call for estimates with stated assumptions, substitute measures when the desired ones are unavailable, or recovery of supported claims from poorly formatted outputs. In BCP, the initial strategy allows the final Mission to search for a missing link; the final GLM strategy states, "Treat upstream 'partial' status as informative but not blocking," and specifies queries to try next. Qwen also checks which earlier claims remain valid when a candidate changes. The common change is a next action matched to the problem that remains unresolved.

Across settings, the policies clarify what work is needed, who should do it, how to check the result, and what to do when information is incomplete.

D LIMITATIONS

Figure 5 connects the six limitations below to possible extensions of MASCraft.

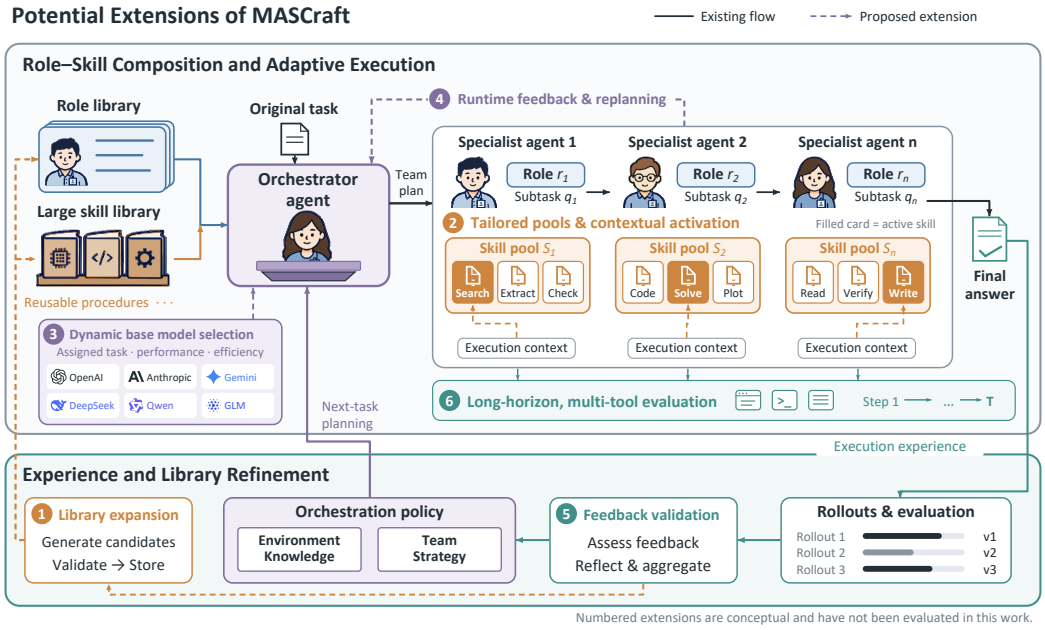


Figure 5: Potential extensions of MASCraft corresponding to limitations (1)–(6). Dashed arrows indicate proposed connections; highlighted skills illustrate contextual activation within tailored pools. Model logos illustrate candidate base models for task-dependent selection.

(1) Library Expansion. The quality and coverage of the role and skill libraries shape the guidance available to specialists. These libraries are prepared offline and remain fixed during evolution and testing. Automatic expansion would require mechanisms to generate, validate, and persist new entries in response to capability gaps encountered during execution.

(2) Skill Composition. Each specialist currently receives one role and one skill per assignment. For much larger skill libraries, a possible extension is to dynamically allocate a tailored skill pool to each specialist based on its role and assigned work, allowing it to activate skills from its own pool according to the evolving execution context.

(3) Base Model Selection. Within each experimental configuration, the orchestrator and all specialist agents share the same base model, even though their assigned tasks may differ in difficulty and capability requirements. The current orchestrator therefore does not adapt model choice to individual assignments. A possible extension is to dynamically select each agent's base model according to its assigned task, balancing task performance with inference cost and latency.

(4) Runtime Adaptation. Team plans remain fixed within each rollout. Although downstream specialists use intermediate results, these results do not trigger changes to task assignments or information dependencies. Runtime replanning could extend adaptation to situations where new evidence changes the required division of labor.

(5) Feedback Dependence. Policy evolution relies on task-level evaluation and LLM-based interpretation of traces. Outcome differences may reflect orchestration choices, skill limitations, or stochastic execution, making diagnosis uncertain. Further work is needed to assess feedback reliability and validate proposed policy revisions, particularly when feedback is sparse or noisy.

(6) Evaluation Scope. Our experiments use controlled tool settings: HLE and DRB have no external tools, while BCP uses local retrieval. They do not cover long-horizon tasks requiring sustained interaction with multiple tools. Evaluation in the broader environment illustrated in the figure is needed to establish the framework's effectiveness and stability in such settings.