INHERIT-MAS: TEST-TIME EVOLUTION OF MULTI-AGENT SYSTEMS THROUGH WORKFLOW AND EXECUTION INHERITANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) built from large language models coordinate specialized agents to tackle complex tasks, but effective workflows are difficult to design in advance. Test-time evolution refines workflows using execution feedback, yet broad revisions can disturb useful components, while re-executing unchanged requests can incur redundant computation. Inspired by the interplay of inheritance and selection in biological evolution, we introduce Inherit-MAS, which makes inheritance explicit at the workflow and execution levels. A meta-model first synthesizes a workflow of worker agents with declared roles, communication inputs, and tool permissions, and a separately prompted judge scores each executed candidate and diagnoses its deficiencies. In ordinary refinement rounds, *workflow inheritance* starts from the latest completed candidate, may discard removable nodes judged unhelpful, and applies a validated edit to address the diagnosed deficiency. When the new candidate executes, *execution inheritance* inherits eligible stored results only if the complete resolved request and execution context match, avoiding redundant model and tool calls. With GPT-4o-mini workers, Inherit-MAS achieves 55.4% completion on WorkBench and 49.7% joint F1 on HotpotQA FullWiki, outperforming EvoAgent, EvoMAS, and TacoMAS. With Qwen3-32B workers, it also exceeds these evolving-MAS baselines on both benchmarks. Compared with rerunning the same controller with execution inheritance disabled, execution inheritance reduces worker-token usage by 29.1% on WorkBench and 34.6% on HotpotQA, and total token usage by 5.3% and 18.1%.

1 INTRODUCTION

Large language model (LLM) agents are increasingly moving beyond single-turn question answering toward complex tasks that require planning, tool use, interaction with external environments (Yao et al., 2023), and adaptation over extended periods of time (Shinn et al., 2023). One approach to managing these demands is to coordinate specialized language-model agents in a multi-agent system (MAS), using workflows for decomposition, tool use, communication, and verification. Rather than asking one LLM to maintain the entire reasoning and execution trajectory, a MAS can divide the task among agents with different responsibilities, execute independent subtasks in parallel, exchange intermediate results, and assign other agents to critique or verify them. Systems such as AutoGen (Wu et al., 2024) and MetaGPT (Hong et al., 2024) instantiate these patterns through programmable conversations and role-based procedures, while automatic design methods search agent programs (Hu et al., 2025), executable workflows (Zhang et al., 2025c), prompts, and graph connections (Zhuge et al., 2024). Across these approaches, performance depends on task decomposition, role and tool assignment, information flow, and verification. Because their interactions are difficult to anticipate, the initial execution may expose weaknesses in evidence collection, tool use, or coordination.

However, such diagnostic information becomes available only after the initial workflow has been synthesized and executed. A fixed workflow can respond to observations within its prescribed execution pattern, but it cannot revise that pattern to address weaknesses exposed by a particular task. Evolving-MAS methods aim to close this gap by revising or modifying an initial system rather than treating it as final, through evolutionary operations over an expert agent (Yuan et al., 2025), pools of promising configurations with accumulated experience (Hu et al., 2026), or coupled fast 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.

slow test-time loops (Xu et al., 2026). Yet deciding what to change is only part of this process. A workflow may retrieve relevant evidence but fail to combine it into a complete answer. Changing the retrieval strategy along with the integration step may disturb work that was already useful. Conversely, retaining an agent's configuration does not establish that its previous result remains applicable, since a revision may change the information it receives. Workflow evolution therefore needs to consider both the scope of a revision and its consequences for completed computations.

Inheritance gives evolution a memory, while natural selection favors beneficial variations and acts against harmful ones (Darwin, 1859). This interplay suggests that test-time MAS evolution should use execution feedback to guide change without unnecessarily losing useful work from earlier attempts. An unsuccessful attempt can still be informative, as its trace may reveal useful partial results alongside missing evidence, redundant work, or coordination failures. Such feedback offers a basis for cumulative adaptation, in which each attempt builds on what earlier executions have revealed. The aim is not to preserve every earlier choice, since some choices must be revised or abandoned, but to use this experience to guide subsequent changes. Under a finite test-time budget, both the quality of those changes and the cost of evaluating them matter. We therefore ask: *How can a MAS use execution feedback to improve task performance while controlling the inference cost of test-time evolution?*

To answer this question, we propose **Inherit-MAS**, a test-time evolution loop with explicit workflow and execution inheritance. A meta-model synthesizes an initial workflow from the task and its interface, specifying agent roles, prompts, tools, and communication links. Each valid candidate is executed, and a separately prompted judge scores its output and trace and provides node-level feedback. In each ordinary refinement round, *workflow inheritance* starts from the latest completed candidate (the incumbent). Guided by the critique, the meta-model may discard removable nodes judged unhelpful, then applies one validated edit intended to address a reported deficiency. The rest of the kept workflow is inherited unchanged. During execution, *execution inheritance* reuses an eligible node's stored result only when its complete resolved request (including upstream results and the execution-context fingerprint) matches a verified record. All other nodes execute live. After the attempt budget is exhausted, the ranking rule returns the highest-ranked completed candidate's stored output.

We evaluate Inherit-MAS on WorkBench (Styles et al., 2024) and HotpotQA FullWiki (Yang et al., 2018). The paper makes three contributions:

- We formulate test-time MAS evolution as explicit workflow inheritance. After initial synthesis, each candidate starts from the latest candidate, keeps the nodes that a node-level critique identifies as useful, and changes the kept workflow through one validated edit directed at the diagnosed deficiency.
- We introduce conservative change localization and request-verified execution inheritance, allowing eligible model and read-only tool results to be used directly across related candidates without altering the candidate being evaluated.
- On WorkBench and HotpotQA FullWiki benchmarks, Inherit-MAS outperforms EvoAgent, EvoMAS, and TacoMAS on overall completion and joint F1 scores while using less inference than EvoMAS and TacoMAS. We also show that, compared with independent reruns with execution inheritance disabled, execution inheritance avoids 29.1% and 34.6% of the live worker inference on the two benchmarks and 5.3% and 18.1% of total tokens.

2 Background and Problem Formulation

Multi-Agent Systems. An LLM-based multi-agent system (MAS) coordinates agents with complementary roles to divide a task and exchange intermediate results. Figure 1 illustrates one workflow in which a planner guides workers that interact with the environment, an integrator combines their findings, and a verifier checks the result before the final output is produced.

Given a task x and an interface $\mathcal{I}$ specifying allowed roles, tools, input/output formats, and the execution environment, a workflow-based MAS can be represented by a directed graph $G = (V, E)$. Here V is the node set and E is the set of directed edges carrying intermediate results between nodes. A node $v \in V$ is an agent or workflow operation with a role, prompt, tool set, decoding configuration, and declared inputs and outputs. Executing G yields an output y, an observable trace τ of messages and tool interactions, and a cost ledger c. Task performance is measured by a utility function U, which assigns a score $U(y)$ to the answer or resulting environment state.

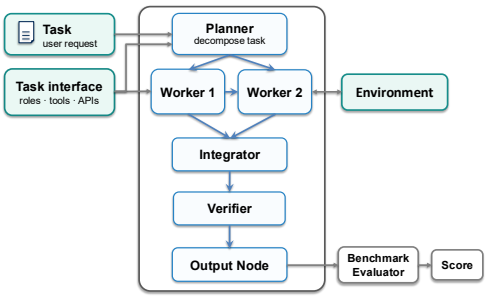

Figure 1: Illustrative multi-agent workflow. The task, its interface, and the environment (green) are external to the workflow (outlined box). The benchmark evaluator scores the output.

Evolving Multi-Agent Systems. An evolving MAS uses execution feedback to adapt agent configurations and interactions rather than keeping its workflow fixed. For task-specific evolution, let K be the number of candidate attempts and let $k \in \{0, \ldots, K-1\}$ index them in construction order. The workflow proposed at index k is denoted by G_k. Executing a valid G_k yields its output y_k, trace τ_k, and cost ledger c_k. Later candidates may revise an earlier workflow or recombine components from a population, using feedback such as execution outcomes, environment rewards, or external evaluations. The final output may be selected from one candidate or assembled from several candidates' outputs. In the single-candidate case, k^* denotes the selected candidate's index and y_{k^*} its output. Evolution aims to improve task performance under the available computational and interaction budgets.

3 Related Work

Multi-Agent Systems. Multi-agent systems coordinate role-specialized language-model agents through communication, tool use, and structured handoffs. ReAct (Yao et al., 2023) established an action-observation loop for tool-using language models, CAMEL (Li et al., 2023) studied cooperation through role-playing agents, AutoGen (Wu et al., 2024) made inter-agent conversations programmable, and MetaGPT (Hong et al., 2024) organized specialized roles around procedural workflows. Automatic design methods search agent programs through meta-search (Hu et al., 2025), optimize graph prompts and connectivity (Zhuge et al., 2024), and construct natural-language or executable workflows (Li et al., 2024; Zhang et al., 2025c). AgentSquare (Shang et al., 2025) searches modular agent designs. DyLAN (Liu et al., 2024) and G-Designer (Zhang et al., 2025b) adapt team composition or communication topology, while DarkForest (Li et al., 2026) passes calibrated, policy-approved evidence to a coordinator. MaAS (Zhang et al., 2025a), FlowReasoner (Gao et al., 2025), and MAS-GPT (Ye et al., 2025) construct query-dependent systems. These methods establish the roles, tools, prompts, and graph structures that an evolving MAS can refine.

Evolving Multi-Agent Systems. Evolving MAS methods use execution feedback to revise agent configurations and communication structures. EvoAgent (Yuan et al., 2025) expands an initial expert into a multi-agent system through evolutionary operations. EvoMAS (Hu et al., 2026) evolves configurations through mutation and crossover while retaining promising configurations and accumulated experience. TacoMAS (Xu et al., 2026) couples fast updates to agent capabilities with slower changes to the agent population and communication graph. Agentic Neural Network (Ma et al., 2026) refines layered agent teams from textual feedback, and Evoflux (Bhandari et al., 2026) repairs tool-workflow graphs through schema-validated edits. Skill-centric approaches evolve reusable orchestration knowledge and execution procedures (Lin et al., 2026; Pan et al., 2026). However, workflow updates can span several components and affect downstream computations. Retaining configurations or prior outputs alone does not establish which stored results remain valid after such changes. Inherit-MAS combines explicit selection of workflow components to retain, validated refinement of the retained workflow, and inheritance of stored execution results when the complete request and execution context match. Appendix A provides detailed comparisons with workflow design, evolution, and execution-reuse methods.

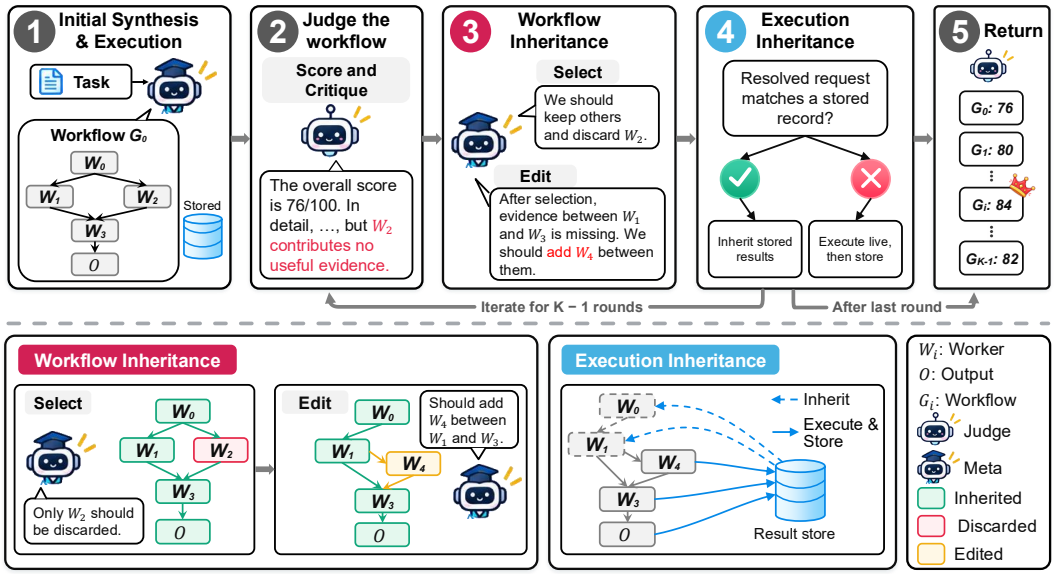

Figure 2: Overview of Inherit-MAS. Top: the ordinary path from initial synthesis and execution (1) to judgment (2), workflow inheritance (3), and execution inheritance (4), followed by final candidate ranking (5). The $K-1$ refinement rounds shown assume a successful initial attempt and omit invalid attempts and synthesis fallbacks. Bottom left: Select may discard removable nodes judged unhelpful; Edit changes the kept workflow within one declared scope. Bottom right: an eligible node inherits a result only when its complete resolved request matches a verified record. Otherwise, it executes live and stores its result if eligible.

4 METHOD

4.1 OVERVIEW

Inherit-MAS searches over directed acyclic workflows $G_k = (V_k, E_k)$ within a budget of K candidate attempts. Figure 2 shows the ordinary execution path. Step 1 synthesizes a valid initial workflow and executes it, storing eligible node results in an initially empty task-local result store. Step 2 uses a separately prompted judge to assess the output and trace and provide node-level feedback. In step 3, *workflow inheritance* refines the latest completed candidate (the incumbent): *Select* may discard removable nodes judged unhelpful, and *Edit* applies one validated change to the kept workflow, intended to address a reported deficiency. Step 4 executes the new candidate with *execution inheritance*: eligible nodes inherit stored results only when their complete resolved requests match verified records; all other nodes execute live. Execution is followed by judgment before the next attempt. After the attempt budget is exhausted, step 5 returns the output of the highest-ranked completed candidate. Invalid attempts and synthesis fallbacks are handled as specified below.

4.2 INITIAL SYNTHESIS AND JUDGMENT

The first attempt has no incumbent: the meta-model's synthesis procedure $\mathcal{S}$ proposes $G_0 = \mathcal{S}(x, \mathcal{I})$ from the task and its interface. A valid workflow is executed to produce (y_k, τ_k, c_k). The separately prompted judge $\mathcal{J}$ then returns $(q_k, \gamma_k) = \mathcal{J}(x, G_k, y_k, \tau_k)$, where $q_k = (q_k^{\mathrm{cmp}}, q_k^{\mathrm{cov}}, q_k^{\mathrm{grd}}, q_k^{\mathrm{all}}) \in [0, 100]^4$ contains completion likelihood, requirement coverage, artifact grounding, and overall quality, and γ_k contains the textual critique and keep/fix recommendations. A candidate is *completed* once its execution and a valid judge response have been recorded. Because the judge receives the workflow and recorded node outputs, it is instructed to identify useful and unhelpful contributions when the trace supports such attribution (step 2 in Figure 2).

4.3 WORKFLOW INHERITANCE

Workflow inheritance governs ordinary refinement from the most recently executed and judged candidate. It has two stages: *Select* chooses which nodes to retain, and *Edit* modifies the kept workflow. Select may discard several nodes, so the one-edit restriction applies to the kept workflow, not to the entire difference between parent and child. Candidate-level ranking is a separate operation described in Section 4.5.

Select: inherit or discard nodes. In an ordinary refinement attempt $k > 0$, let $p(k) < k$ denote the index of the latest completed candidate, whose workflow $G_{p(k)}$ is the incumbent. Let $\mathcal{R}(G)$ contain the nodes whose individual removal leaves G valid under $\mathcal{I}$. The output node and nodes needed to satisfy required input constraints are excluded. The meta-model's node selector $\mathcal{P}_{\text{sel}}$ is instructed to discard the node only when the feedback or recorded output indicates that it contributes nothing useful or harms the result. It proposes a discard set Δ_k^-, and pruning removes those nodes and their incident edges:

$$\begin{aligned} \Delta_k^- &= \mathcal{P}_{\text{sel}}\big(x, G_{p(k)}, \tau_{p(k)}, q_{p(k)}, \gamma_{p(k)}\big)\,, \\ G'_k &= \mathsf{Prune}(G_{p(k)}, \Delta_k^-). \end{aligned} \tag{1}$$

The selection is accepted only if $\Delta_k^- \subseteq \mathcal{R}(G_{p(k)})$ and $\mathsf{Valid}(G'_k, \mathcal{I}) = 1$, where Valid is the deterministic workflow validator. Checking the combined discard set is necessary because individually removable nodes need not be jointly removable. If $\mathcal{R}(G_{p(k)}) = \varnothing$, no node can be removed individually while preserving validity, so selection is skipped and $G'_k = G_{p(k)}$. Every kept node retains its role, prompt, assigned subtask, tools, and remaining inputs.

Edit: propose and validate. Let $\mathcal{A}(G)$ denote the admissible edits for graph G, $\mathcal{P}$ the proposer, and $\mathcal{T}$ the deterministic edit applicator. The implementation exposes a finite menu of operations and targets. The proposer chooses one entry and supplies any required replacement text or node instructions. It receives the incumbent's execution trace and judge feedback but edits the kept workflow:

$$\begin{aligned} a_k &= \mathcal{P}\big(x, G'_k, \tau_{p(k)}, q_{p(k)}, \gamma_{p(k)}, \mathcal{A}(G'_k)\big)\,, \\ G_k &= \mathcal{T}(G'_k, a_k), \quad a_k \in \mathcal{A}(G'_k), \\ &\quad \mathsf{Valid}(G_k, \mathcal{I}) = 1. \end{aligned} \tag{2}$$

The transition is accepted only after validation. Cycles, incompatible input/output connections, unauthorized tools, and other violations of $\mathcal{I}$ are rejected. $\mathcal{T}$ changes only fields within the selected edit's declared scope, preserving the rest of the kept workflow. This guarantees structural validity and scoped modification, not that the edit repairs the diagnosed problem or improves task performance. Unsuccessful attempts consume a candidate slot and remain in failure and cost accounting. The bounded response-repair procedure is specified in Appendix C.1.

An edit may change a node's prompt, assigned subtask, or tool access; add or remove a node or edge; reorder inputs; or insert a node between connected nodes, as with W_4 between W_1 and W_3 in Figure 2. The insertion adds a path through the new node while retaining the original edge. Appendix C.3 gives the edit family and validator checks. When no candidate has yet completed, the next attempt repeats initial synthesis instead of refinement. With completed candidates available, the final attempt uses workflow resynthesis if none has a valid output, or if the highest overall-quality score q^{all} is at most one point above that of the first completed candidate. This restart prompt receives the highest-ranked prior workflow and its judge feedback and requests a meaningfully different valid workflow. It does not require component preservation and is treated as a transition without workflow inheritance. Algorithm 1 gives the complete controller.

4.4 EXECUTION INHERITANCE

Change localization identifies where requests may differ, while execution inheritance decides whether a previous result can be used. A hit requires an eligible node, an exact match of the complete resolved request, and a record that passes integrity verification. A node inside the affected region can still inherit a result if its current request matches one stored earlier.

Localize: identify possible downstream effects. After Select and Edit have been validated, localization compares G_k with the original incumbent $G_{p(k)}$, accounting for both pruning and editing. Let $\Delta_k = \mathsf{Touch}(G_{p(k)}, G_k)$ contain every new node and every retained node whose local specification, ordered incoming wiring, or effective execution settings change. This includes any retained node that loses an input through pruning and any node whose tool or retrieval allowance changes (Appendix C.6). The conservative affected region is

$$D_k = \Delta_k \cup \mathrm{Desc}_{G_k}(\Delta_k), \tag{3}$$

where $\mathrm{Desc}_G(S)$ denotes nodes reachable from S by one or more directed edges. For synthesis without a parent, $D_k = V_k$. This partition separates structurally unchanged nodes $V_k \setminus D_k$ from potentially affected nodes D_k. It does not imply that the behavioral effect is small, nor does it authorize inheritance. Both regions require the complete request verification described next.

Verify and inherit. For node $v \in V_k$, the *resolved request* contains everything needed to execute it, including its current specification, ordered upstream records, and execution context. We form the request and its lookup key as follows.

$$\begin{aligned} \phi_k &= \mathsf{Fingerprint}(e_k), \\ r_{k,v} &= \mathsf{Resolve}\Big(x, v, [(\pi_i, w_{k,u_i})]_{i=1}^{d_v}, \phi_k\Big), \\ \kappa_{k,v} &= h(r_{k,v}). \end{aligned} \tag{4}$$

Here d_v is the number of upstream inputs. Each input has a label π_i and a record w_{k,u_i} from source node u_i. These records include outputs, metadata, and exposed tool observations or errors. Resolve includes the actual messages, model and decoding settings, allowed tools, and execution limits. The context e_k covers the runtime and parser, initial environment state, and fixed tool or retrieval service. Its fingerprint ϕ_k reflects their contents and versions, regardless of which environment copy is used. The hash h applies SHA-256 to the canonically serialized request. The node's own tool observations arise after lookup and are stored with its result, not included in the key.

The task-local store $\mathcal{M}$ maps request keys to records containing output o, tool observations and artifacts A, original token usage t, and execution errors ϵ. All candidates share this store. A node inherits a result only if it is eligible and a matching record passes verification,

$$z_{k,v} = \mathbf{1}\{\mathsf{Eligible}(v, e_k) \ \wedge \ \mathsf{ValidRecord}(\mathcal{M}, \kappa_{k,v}) = 1\}, \tag{5}$$

where Eligible checks whether inheritance is permitted. ValidRecord checks the record's format, request key, and content hash. It returns one for a verified match and zero when no record exists. A malformed or inconsistent record stops execution instead of being treated as a miss.

Language-model nodes without state-changing tools and deterministic read-only operations can be eligible. State-changing operations are never eligible. For stateful tasks, each candidate starts from a fresh copy of the same initial environment. Intermediate nodes can only read it, and a designated sink performs every state-changing action live. After verification, the node either inherits the stored result or executes live,

$$(o, A, t, \epsilon)_{k,v} = \begin{cases} \mathcal{M}[\kappa_{k,v}], & z_{k,v} = 1, \\ \mathsf{Exec}(v, r_{k,v}, e_k), & z_{k,v} = 0. \end{cases} \tag{6}$$

Eligible live results are stored without later modification. The downstream record $w_{k,v}$ contains the node's metadata, output, exposed artifacts, and errors, but not token usage. Inherited results restore the same evidence for downstream nodes and the judge. Inherited errors remain errors.

A hit uses no live worker tokens. Its original usage is recorded separately as an inherited token equivalent. Synthesis, selection, refinement, and judging remain live. Inheritance restores a recorded result without assuming that a fresh remote-model call would reproduce it. Appendices E.3 and E.4 give the accounting details and deterministic-runtime checks.

4.5 Ranking and Return

Let $\mathcal{C} \subseteq \{0, \ldots, K-1\}$ be the indices of completed candidates. For $k \in \mathcal{C}$, let b_k count recorded node execution errors plus one if the final output is invalid under the benchmark adapter. The online

ranking key and returned candidate are

$$s_k = \left(-b_k, \mathbf{1}\{y_k \text{ is valid}\}, q_k^{\text{all}}, q_k^{\text{cov}}, q_k^{\text{grd}}, -k\right), \qquad k^* = \arg\operatorname*{prio}_{i \in \mathcal{C}} \max s_i. \tag{7}$$

Here $\mathbf{1}$ is the binary indicator, and priority comparison proceeds from left to right; the first differing component determines the preferred candidate. A tie on all preceding components favors the earlier candidate. Cost is excluded from the key. If $\mathcal{C}$ is empty, no candidate output is returned and the task is recorded as a failure. Otherwise, step 5 returns y_{k^*} without re-executing the workflow. Retaining all completed candidates prevents a lower-ranked child from displacing its parent in the final selection, but this ordering does not guarantee improvement in the unavailable benchmark utility U.

Table 1: WorkBench completion percentages by worker backbone and domain. Bold marks the best MAS result within each backbone, excluding the Single ReAct reference. The shaded Overall column is the primary metric, computed over all evaluation tasks (Appendix D).

System	Analytics	Calendar	CRM	Email	Project mgmt.	Multi-domain	**Overall**
Workers: GPT-4o-mini							
Single ReAct (reference)	35.0	50.0	20.0	35.0	20.0	60.0	41.5
EvoAgent	55.0	40.0	**20.0**	25.0	20.0	35.0	33.8
EvoMAS	25.0	35.0	**20.0**	35.0	20.0	37.5	30.8
TacoMAS	20.0	55.0	**20.0**	5.0	20.0	57.5	34.6
Inherit-MAS	**60.0**	**75.0**	**20.0**	**45.0**	**35.0**	**67.5**	**55.4**
Workers: Qwen3-32B							
Single ReAct (reference)	65.0	55.0	20.0	35.0	25.0	55.0	46.2
EvoAgent	25.0	30.0	20.0	20.0	20.0	40.0	28.5
EvoMAS	20.0	35.0	20.0	30.0	15.0	**52.5**	33.1
TacoMAS	35.0	**60.0**	**40.0**	0.0	**25.0**	42.5	34.6
Inherit-MAS	**75.0**	50.0	20.0	**40.0**	20.0	**52.5**	**46.2**

5 Experiments

5.1 Experimental Setup

Baselines. We compare our proposed **Inherit-MAS** with four baselines. **Single ReAct** (Yao et al., 2023) is one agent with the complete public tool interface and no meta-model, evolution, or execution inheritance. We report it as a non-MAS reference. **EvoAgent** (Yuan et al., 2025) grows a population from an initial expert agent. **EvoMAS** (Hu et al., 2026) evolves structured MAS configurations with execution traces and persistent cross-task experience. **TacoMAS** (Xu et al., 2026) couples fast capability updates with slower topology evolution.

Benchmarks. We use two benchmarks for the evaluation: **WorkBench** (Styles et al., 2024) and **HotpotQA** (Yang et al., 2018). WorkBench contains workplace tasks over email, calendar, project management, CRM, and analytics databases, with outcome-centric evaluation of the final state. HotpotQA requires multi-document reasoning and sentence-level supporting facts. Inherit-MAS, Single ReAct, TacoMAS, and the Qwen EvoMAS row share a budget of four calls per candidate to the same fixed BM25 index, returning five documents per call and allocated across a candidate's retrieval nodes; EvoAgent and the GPT-4o-mini EvoMAS row receive four calls per agent (Appendix D.1).

Implementation. Inherit-MAS uses GPT-4o-mini workers (OpenAI, 2024) and GPT-5.4-mini (OpenAI, 2026) for synthesis/refinement and the separately prompted judge, all at temperature zero. Each trajectory contains an initial workflow and four subsequent candidates, with one Select decision and one validated edit ordinarily applied per refinement round. Single ReAct uses the same worker model. For WorkBench, every candidate runs from a fresh environment copy and a deterministic executor applies its proposed write batch.

Table 2: HotpotQA FullWiki F1 scores (%) by worker backbone. Bold marks the best MAS result within each backbone. SP F1 is supporting-fact F1. The shaded Joint F1 column is the primary metric, the official combination of answer and supporting-fact scores (Appendix D).

System	Answer F1	SP F1	**Joint F1**
Workers: GPT-4o-mini			
Single ReAct (reference)	69.5	61.9	46.0
EvoAgent	69.8	47.4	34.8
EvoMAS	64.5	54.7	39.2
TacoMAS	66.8	**64.8**	47.8
Inherit-MAS	**73.3**	61.7	**49.7**
Workers: Qwen3-32B			
Single ReAct (reference)	63.0	51.7	37.1
EvoAgent	61.5	45.7	32.3
EvoMAS	61.1	56.4	39.4
TacoMAS	56.8	**60.0**	39.9
Inherit-MAS	**65.3**	58.5	**42.7**

All MAS baselines use released implementations with benchmark-specific adaptations. EvoAgent uses GPT-4o-mini; EvoMAS and TacoMAS use GPT-4o-mini workers with GPT-5.4-mini meta-models and judges. The second backbone configuration uses Qwen3-32B (Yang et al., 2025) workers in non-thinking mode for all five systems. Meta-models and judges remain GPT-5.4-mini, while EvoAgent uses Qwen3-32B throughout. Appendices D.1 and D.2 provide the code versions, benchmark adaptations, model-routing details, and decoding settings.

Evaluation. All systems are evaluated on the same fixed set of tasks, and the code and configurations of every system were fixed before the final runs. During a run, the meta-model and judge never observe reference actions, answers, supporting facts, or official metrics, which are used only for post-run scoring (Appendix C.7). Failures and invalid outputs remain in the denominator. On WorkBench, the benchmark evaluator scores the final environment state, and the primary metric is official completion. On HotpotQA, the primary metric is official joint F1, which requires the returned answer together with sentence-level supporting facts in the required structured form, and we also report answer and supporting-fact F1. For cost, every run records the token usage of every model call, which we split into worker inference and meta-model and judge inference for synthesis, node selection, refinement, and judging, and inherited results are reported separately as token equivalents. External systems run under their documented configurations.

5.2 Main Results

Inherit-MAS outperforms the evolving-MAS baselines on both benchmarks with both worker backbones (Tables 1 and 2). On WorkBench with GPT-4o-mini workers, it is 13.8 points above Single ReAct and 20.8 points above TacoMAS (the best evolving-MAS baseline), while every evolving-MAS baseline trails Single ReAct. With Qwen3-32B workers, it matches Single ReAct at 46.2% and exceeds the best evolving-MAS baseline by 11.5 points. On HotpotQA, joint F1 couples the answer with sentence-level citations. With GPT-4o-mini workers, Inherit-MAS is 3.7 points above Single ReAct and 1.9 points above TacoMAS, the closest competitor. It has the highest answer F1 of all systems, while its supporting-fact F1 stays at Single ReAct's level, and TacoMAS has the best supporting facts but weaker answers. The same pattern holds with Qwen3-32B workers, where Inherit-MAS is 5.6 points above Single ReAct and 2.8 points above TacoMAS, again with the highest answer F1, while TacoMAS retains the best supporting facts.

Across the four benchmark–backbone settings, Inherit-MAS is the only evolving-MAS system that matches or exceeds Single ReAct in every setting, and its margin over the other evolving-MAS systems is widest on WorkBench, where completion is scored on the final environment state. In each ordinary refinement round, Inherit-MAS inherits the nodes of the latest candidate that the judge finds

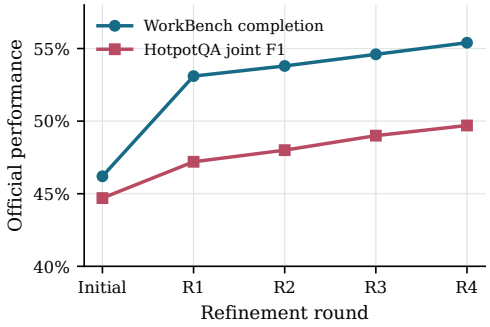

Figure 3: Official performance of the returned workflow after initial synthesis (Initial) and after each refinement round (R1 to R4, counted by candidate index). The R4 points are the reported final results.

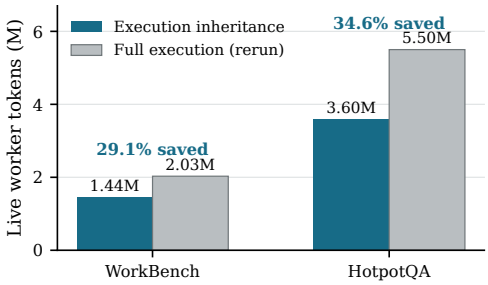

Figure 4: Live worker tokens with execution inheritance (original runs) and with execution inheritance disabled (independent reruns of the same controller on the same tasks, averaged over two reruns on WorkBench).

useful and changes the kept workflow by the validated edit. Inherit-MAS reaches these scores with fewer tokens than EvoMAS and TacoMAS, which use 16.8× and 22.9× its total on WorkBench and 13.7× and 2.1× on HotpotQA under each system's own configuration (Appendix E).

5.3 Performance across Refinement Rounds

After each task's evolution finishes, we score offline the workflow that Inherit-MAS would have returned after initial synthesis (Initial) and after each refinement round (R1 to R4), using the official metric (Figure 3). Both curves rise at every round, by 5.0 points in total on HotpotQA and 9.2 points on WorkBench, where the first round contributes 6.9 points. These gains show that the evolution process yields progressively better performance.

5.4 Execution Inheritance and Token Cost

We rerun Inherit-MAS on the same tasks with execution inheritance disabled, keeping the controller, judge, edit menu, and retrieval budget unchanged. Every node executes live, and the reruns are independent trajectories.

From Figure 4, full execution spends 2.03M worker tokens on WorkBench and 5.50M on HotpotQA, against 1.44M and 3.60M with execution inheritance, so execution inheritance saves 29.1% and 34.6% of worker tokens. Meta-model and judge inference stays live in both settings, so total token savings are 5.3% and 18.1%, smaller on WorkBench where worker calls are a smaller share of the budget. The savings come from requests that an edit leaves unchanged, so within the original trajectories they are concentrated in prompt edits, which inherit 47.3% and 75.0% of worker-token equivalents (Table 3). Full execution achieves 55.4% completion in the WorkBench rerun, the same as execution inheritance.

6 Conclusion

Inherit-MAS makes inheritance explicit in test-time multi-agent system evolution by keeping an incumbent's useful nodes, applying one validated, diagnosis-targeted edit to the kept workflow, and inheriting stored results for unchanged requests. On WorkBench and HotpotQA benchmarks, it achieves higher primary-metric scores than the evolving-MAS baselines with both GPT-4o-mini and Qwen3-32B workers. In the GPT-4o-mini experiments, execution inheritance reduces worker-token usage by 29.1% and 34.6% relative to independent full-execution reruns, corresponding to total token savings of 5.3% and 18.1%.

AI USE STATEMENT

In this work, we used generative AI tools to help develop conceptual frameworks, to formulate mathematical claims, to provide feedback on the research methodology and experiments, and to implement methods. We have not used generative AI tools to generate synthetic data sets, to propose or refine hypotheses, to clean and reformat datasets, or to interpret results, and the remaining tasks with required disclosure (providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, and supporting qualitative and thematic data analysis) are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, to create and modify scientific figures, to create artifacts, to draft parts of this paper, to summarize existing literature and identify relevant literature, to edit the paper for readability, and to propose a title and keywords. We have reviewed all AI-assisted work. AI-generated code was verified with automated tests and comparisons against the official benchmark evaluators, every reported value was checked against saved run records, AI-drafted text and literature summaries were checked against the cited sources and revised by the authors, and the experiments use existing benchmark data rather than AI-generated datasets. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The method is specified in Section 4, and the implementation settings, prompts, workflow and output rules, evaluation protocols, and runtime limits appear in Section 5.1 and Appendix C. Baseline code versions, adapters, and model assignments are documented in Appendix D.1, and token usage is reported in Appendix E.2. The source code of implementation, evaluation task lists, and scoring scripts are available anonymously at https://anonymous.4open.science/r/Inherit-MAS-ICLR2027-8202.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 81–95, 2024. URL https://proceedings.mlr.press/v235/abhyankar24a.html.

Kushal Raj Bhandari, Ling Yue, Ching-Yun Ko, Dhaval Patel, Shaowu Pan, Pin-Yu Chen, and Jianxi Gao. Evoflux: Inference-time evolution of executable tool workflows for compact agents. *arXiv preprint arXiv:2606.12674*, 2026. URL https://arxiv.org/abs/2606.12674.

Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, first edition, 1859. URL https://darwin-online.org.uk/converted/published/1859_Origin_F373/1859_Origin_F373.html.

Faisal Fareed. Cost-aware speculative execution for LLM-agent workflows: An integrated five-dimension method. *arXiv preprint arXiv:2606.07846*, 2026. URL https://arxiv.org/abs/2606.07846.

Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. FlowReasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025. URL https://arxiv.org/abs/2504.15257.

Matthew A. Hammer, Jana Dunfield, Kyle Headley, Nicholas Labich, Jeffrey S. Foster, Michael Hicks, and David Van Horn. Incremental computation with names. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 748–766, 2015. doi: 10.1145/2814270.2814305.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, 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*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

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*, 2026. URL https://arxiv.org/abs/2602.06511.

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*, volume 36, 2023. URL https://arxiv.org/abs/2303.17760.

Yi Li, Songtao Wei, Dongming Jiang, Zhichun Guo, Qiannan Li, and Bingzhe Li. DarkForest: Less talk, higher accuracy for multi-agent LLMs. *arXiv preprint arXiv:2605.25188*, 2026. URL https://arxiv.org/abs/2605.25188.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. AutoFlow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024. URL https://arxiv.org/abs/2407.12821.

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

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Xiaowen Ma, Yunpu Ma, Chenyang Lin, Sikuan Yan, Jinhe Bi, Zixuan Cao, Yijun Tian, Volker Tresp, and Hinrich Schuetze. Self-evolving multi-agent systems via textual backpropagation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 9918–9951, 2026. doi: 10.18653/v1/2026.findings-acl.483. URL https://aclanthology.org/2026.findings-acl.483/.

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*, 2023. URL https://arxiv.org/abs/2303.17651.

Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. Build systems à la carte. *Proceedings of the ACM on Programming Languages*, 2(ICFP):79:1–79:29, 2018. doi: 10.1145/3236774.

Azam Nouri. StepCache: Step-level reuse with lightweight verification and selective patching for LLM serving. *arXiv preprint arXiv:2603.28795*, 2026. URL https://arxiv.org/abs/2603.28795.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. OpenAI product announcement, 2024.

OpenAI. Introducing GPT-5.4 mini and nano. OpenAI product announcement, 2026.

Shuai Pan, Yixiang Liu, Jiaye Gao, Te Gao, Weiwen Liu, Jianghao Lin, Zhihui Fu, Jun Wang, Weinan Zhang, and Yong Yu. SkillMAS: Skill co-evolution with LLM-based multi-agent system. *arXiv preprint arXiv:2605.09341*, 2026. URL https://arxiv.org/abs/2605.09341.

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*, 2025. URL https://arxiv.org/abs/2410.06153.

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*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: a benchmark dataset for agents in a realistic workplace setting. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=4HNAwZFDcH.

Noppanat Wadlom, Junyi Shen, and Yao Lu. Efficient LLM serving for agentic workflows: A data systems perspective. *Proceedings of the ACM on Management of Data*, 4(3):1–29, 2026. doi: 10.1145/3802046.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

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. URL https://arxiv.org/abs/2605.09539.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. doi: 10.18653/v1/D18-1259.

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*, 2023. URL https://arxiv.org/abs/2210.03629.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. MAS-GPT: Training LLMs to build LLM-based multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72063–72090, 2025. URL https://proceedings.mlr.press/v267/ye25g.html.

Lingzhi Yuan, Chenghao Deng, Fangxu Yu, Souradip Chakraborty, Mohammad Rostami, and Furong Huang. FlowBank: Query-adaptive agentic workflows optimization through precompute-and-reuse. *arXiv preprint arXiv:2606.11290*, 2026. URL https://arxiv.org/abs/2606.11290.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, 2025. doi: 10.18653/v1/2025.naacl-long.315. URL https://aclanthology.org/2025.naacl-long.315/.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852, 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-Designer: Architecting multi-agent communication topologies via graph neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692, 2025b. URL https://proceedings.mlr.press/v267/zhang25cu.html.

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*, 2025c. URL https://openreview.net/forum?id=z5uVAKwmjf.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025d. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9549f7d06700f0966d5f938f1d11022a-Abstract-Conference.html.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

A Extended Related Work

A.1 Workflow Design and Optimization

Automatic design methods search agent programs (Hu et al., 2025), graph prompts and connections (Zhuge et al., 2024), or modular agent designs (Shang et al., 2025). They differ in both the representation being optimized and the setting in which optimization occurs. In its main benchmark experiments, AFlow (Zhang et al., 2025c) optimizes candidate workflows on a validation split before evaluating the selected workflow on held-out test data. Its Appendix F also explores single-question optimization with an LLM judge and no reference answers. Query-dependent generation methods such as MaAS (Zhang et al., 2025a), FlowReasoner (Gao et al., 2025), and MAS-GPT (Ye et al., 2025) further connect workflow construction to individual requests. Inherit-MAS focuses on how an executed task-specific MAS supplies the components and completed computation for its next refinement.

A.2 Workflow Evolution

Iterative feedback provides a basis for refinement at several levels. Reflexion (Shinn et al., 2023) carries verbal feedback across trials, and Self-Refine (Madaan et al., 2023) alternates feedback and response revision. Self-consistency (Wang et al., 2023) instead aggregates independently sampled reasoning paths, while test-time compute scaling studies both search and adaptive response refinement (Snell et al., 2025). Inherit-MAS applies feedback to an executable multi-agent workflow and retains earlier candidates for final selection.

Evolving MAS methods carry prior work forward in different forms. EvoAgent (Yuan et al., 2025) executes newly generated agents and integrates their outputs with the previous result. EvoMAS (Hu et al., 2026) retains pools of promising configurations and accumulated search experience, mutates one component type that may span several agents, and recombines parent configurations through crossover. TacoMAS (Xu et al., 2026) carries capability state and trajectory-derived feedback across fast and slow loops, with slow updates able to combine changes to the population and communication graph. Retaining such information is distinct from verifying whether an individual stored node

result remains valid after a revision. Inherit-MAS makes that decision explicit alongside workflow refinement. Select keeps the incumbent's useful nodes, Edit preserves the kept workflow outside its declared scope, and execution inheritance verifies the complete resolved request and execution context before restoring a result.

Evoflux (Bhandari et al., 2026) is a closely related inference-time method that evolves tool-workflow graphs from execution feedback without updating model weights. It focuses on the execution feasibility of compact planners against live tool servers, whereas Inherit-MAS combines multi-agent workflow refinement with request-verified inheritance of agent outputs.

A.3 Execution Reuse and Caching

Execution reuse differs in what is stored, how a match is established, and whether reuse changes the produced result. Demand-driven incremental computation (Hammer et al., 2015) and build systems (Mokhov et al., 2018) track dependencies to determine when previous computation remains valid. Inherit-MAS draws on this principle when checking node requests after a workflow edit. At the inference-runtime level, InferCept (Abhyankar et al., 2024) addresses the cost of reconstructing prior context around tool interruptions.

Agent systems reuse several kinds of artifacts. StepCache (Nouri, 2026) retrieves a similar cached request, verifies the stored response's steps, and regenerates failing regions. Agentic Plan Caching (Zhang et al., 2025d) adapts plan templates across tasks, and FlowBank (Yuan et al., 2026) precomputes workflows for query-adaptive selection. Helium (Wadlom et al., 2026) combines cached-operator substitution with common-subgraph elimination. Cost-aware speculative execution (Fareed, 2026) makes commit barriers and the admissibility of environment-changing actions explicit.

Inherit-MAS combines workflow evolution with task-local inheritance of eligible node outputs, tool observations, and artifacts under a complete resolved-request and execution-context match. Unlike regenerating a cached response's failing regions, an inheritance hit restores the recorded result unchanged, and Figure 4 measures the execution it avoids against independent reruns of the same controller with inheritance disabled. Whether a fresh model invocation reproduces the stored output is a separate question examined under the deterministic-runtime conditions in Appendix E.4.

B Additional Ablations and Analysis

B.1 Ranking Headroom and Failure Modes

The online ranking does not merely return the last round: it recovers part, but not all, of the variation created by evolution. On WorkBench, the oracle over stored candidates solves 82 tasks versus 72 returned. On HotpotQA it reaches 56.3% joint F1 versus 49.7% returned. This gap motivates better calibrated judges and ensemble ranking, but it also cautions against reporting oracle-any-round as an online system.

The same sealed trajectories (run records that are never modified after a run) also separate initialization, the last generated candidate, and the online ranking without additional model calls. On WorkBench these return policies achieve 49.2%, 50.0%, and 55.4% completion, respectively; on HotpotQA they achieve 44.7%, 44.4%, and 49.7% joint F1. Later candidates are therefore not uniformly better. Refinement creates useful alternatives, while retaining and ranking earlier workflows is important to final performance. The last-candidate view changes only the final return rule; every refinement still builds on the latest candidate.

The stored trajectories provide additional context for the returned-workflow curves in Figure 3. Raw WorkBench candidate completion is not monotonic. It is 46.2, 51.5, 48.5, 42.3, and 50.0% across candidate indices. The 46.2% at index 0 counts the six tasks whose initial proposal was invalid as failures, whereas the 49.2% initialization figure above scores each task's first successfully executed initial workflow, which for those six tasks is the replacement at index 1, four of which succeed. HotpotQA shows the same nonmonotonic pattern, with joint F1 of 44.7, 46.2, 47.4, 45.9, and 44.4%. Evolution should therefore be read as dependent iterative search rather than deterministic hill climbing. Applying the same online ranking rule to each prefix raises the returned candidate to 55.4% and 49.7%, and the returned candidate's score never decreases on either benchmark. The

Table 3: Post-hoc incumbent-to-candidate transition aggregates on the sealed trajectories of the reported Inherit-MAS runs. WorkBench utility is binary completion. HotpotQA utility is joint F1. Improve is the fraction with strictly positive utility delta. Affected is the mean fraction of nodes in the affected region. Inherit. is the inherited share of worker-token equivalents, and the final column is net $\sum \Delta U$ per million live worker tokens. Topology covers node insertions, edge edits, and transitions in which Select discarded nodes. These associations are descriptive, not randomized edit effects.

Benchmark	Edit family	n	Improve	Mean ΔU	Affected	Inherit.	Live tok.	$\sum \Delta U$/M
WB	Restart	73	9.6%	+0.0411	100.0%	2.0%	4,134	+9.94
	Prompt	253	3.6%	+0.0316	70.8%	47.3%	1,679	+18.84
	Subtask	78	1.3%	+0.0000	69.1%	47.2%	1,568	+0.00
	Tool	49	12.2%	−0.0612	100.0%	6.3%	3,272	−18.71
	Topology	13	7.7%	+0.0769	75.0%	44.5%	2,453	+31.36
HP	Restart	184	19.0%	−0.0418	100.0%	0.0%	4,667	−8.95
	Prompt	749	6.8%	−0.0022	52.3%	75.0%	836	−2.68
	Subtask	133	17.3%	+0.0312	91.9%	5.6%	4,101	+7.60
	Tool	8	25.0%	+0.0886	100.0%	8.2%	3,154	+28.10
	Topology	106	21.7%	+0.0344	97.6%	4.4%	3,625	+9.49

offline oracle prefixes rise to 63.1% and 56.3%, showing that the main remaining gap is that the judge does not always identify the best evaluated dependent refinement.

Execution inheritance is concentrated in ordinary refinement rounds. At indices 1 through 3, the inherited share of worker-token equivalents is between 34.7% and 44.0% on WorkBench and between 44.0% and 63.2% on HotpotQA. At index 4 it falls to 17.6% and 21.0%, respectively, because the final-round restart often produces a fresh workflow. The restart's candidate is the returned workflow for 25 WorkBench tasks and 47 HotpotQA questions. The final candidate thus adds substantial oracle headroom but is also the least inheritance-friendly round.

B.2 Edit-Level Outcomes and Locality

Table 3 aggregates 466 WorkBench and 1,180 HotpotQA completed transitions against the incumbent used at proposal time. Only 24 and 134 transitions, respectively, strictly improve official utility. Of those, 8/24 (33.3%) and 37/134 (27.6%) have an affected region that is a strict subset of the workflow. Thus most observed improvements are not local under even the permissive definition that at least one node of the new candidate remains outside the affected region.

Local edits nevertheless provide the clearest cost leverage. Their token-weighted inheritance is 57.5% on WorkBench and 81.8% on HotpotQA, with mean live worker cost of 1,345 and 596 tokens, compared with 4,134 and 4,667 tokens for restarts. Quality does not track that locality: local edits have mean utility deltas of +0.0212 and -0.0007, and the families with positive means differ across benchmarks (prompt and topology edits and restarts on WorkBench; subtask, tool, and topology edits on HotpotQA, where restarts have a negative mean). Prompt edits occupy the useful middle ground, inheriting 47.3% and 75.0% of worker-token equivalents while occasionally improving. The aggregate evidence therefore supports inheritance as a cost mechanism, but not locality as a proxy for edit quality. The latter requires a better gain estimator.

Two deterministically selected post-hoc cases make the capability–inheritance tradeoff concrete. For a WorkBench reassignment task, adding a context edge from one worker to another leaves the outcome unchanged, inherits the unaffected worker's prior result, and re-executes the dependent worker, integrator, and write executor (2,452 inherited worker-token equivalents versus 760 live worker tokens). For a HotpotQA two-hop question, discarding the planner and rewriting the researcher's prompt supplies the missing bridge and raises joint F1 from 0.667 to 1.000, but every node is re-executed (9,614 live worker tokens, none inherited). Local rewiring can preserve prior computation. Useful repairs that reshape the workflow may intentionally invalidate it.

HotpotQA shows a larger win and loss margin (Table 4), although some of the improvement may come from broader retrieval exposure across candidates rather than from topology changes, which we do not separate here.

B.3 PAIRED HEADLINE EFFECTS

Table 4 reports paired differences and task-level win/loss/tie counts for the main-table comparisons. Wins and losses count strictly better and worse official task scores. Ties include equal scores.

Table 4: Inherit-MAS minus comparator on each benchmark's primary metric. WB deltas are percentage points. HP deltas are joint F1.

Bench.	Comparator	Δ	W/L/T
WB	Single ReAct (reference)	+13.8	29/11/90
WB	Initial workflow	+6.2	12/4/114
WB	EvoAgent	+21.5	37/9/84
WB	EvoMAS	+24.6	39/7/84
WB	TacoMAS	+20.8	39/12/79
HP	Single ReAct (reference)	+0.037	92/65/143
HP	Initial workflow	+0.050	54/21/225
HP	EvoAgent	+0.149	125/32/143
HP	EvoMAS	+0.105	110/54/136
HP	TacoMAS	+0.019	81/75/144

The HotpotQA TacoMAS row is the adapted implementation described in Appendix D.1. The released controller, run through a task and output plugin on the same questions, returned prose answers without the structured supporting-fact array that joint F1 requires, so it has no informative paired joint score and is retained only as a released-interface diagnostic.

C INHERIT-MAS IMPLEMENTATION

C.1 COMPLETE CONTROLLER

Algorithm 1 Inherit-MAS controller with execution inheritance

Require: task x, task interface $\mathcal{I}$, cap K, pristine environment e_0

$\mathcal{H} \leftarrow [\,]$; $\mathcal{M} \leftarrow \mathsf{EmptyStore}()$
for $k = 0, \ldots, K-1$ **do**
  $p \leftarrow \perp$
  if $\mathsf{Complete}(\mathcal{H}) = \varnothing$ **then**
    $G_k \leftarrow \mathsf{ValidatedSynthesis}(x, \mathcal{I})$
  else if $k = K-1$ and $\mathsf{Restart}(\mathcal{H})$ **then**
    $G_k \leftarrow \mathsf{ValidatedRestart}(x, \mathcal{I}, \mathcal{H})$ ▷ fresh restart
  else
    $p \leftarrow \mathsf{Latest}(\mathsf{Complete}(\mathcal{H}))$
    $\Delta^- \leftarrow \mathsf{Select}(x, G_p, \tau_p, q_p, \gamma_p, \mathcal{R}(G_p))$; $G' \leftarrow \mathsf{Prune}(G_p, \Delta^-)$ ▷ $\Delta^- = \varnothing$ if $\mathcal{R}(G_p) = \varnothing$
    if G' is invalid **then**
      $\mathcal{H}.\mathsf{Append}(\mathsf{Invalid}(k))$; **continue**
    end if
    $a_k \leftarrow \mathsf{Propose}(x, G', \tau_p, q_p, \gamma_p, \mathcal{A}(G'))$
    $G_k \leftarrow \mathsf{ValidateApply}(G', a_k, \mathcal{I})$
  end if
  if G_k is invalid **then**
    $\mathcal{H}.\mathsf{Append}(\mathsf{Invalid}(k))$; **continue**
  end if
  $D_k \leftarrow V_k$ if $p = \perp$, else $\mathsf{AffectedRegion}(G_p, G_k)$
  $e_k \leftarrow \mathsf{Fork}(e_0)$; $\phi_k \leftarrow \mathsf{Fingerprint}(e_k)$
  for v in a topological order of G_k **do**
    $r_{k,v} \leftarrow \mathsf{Resolve}(x, v, [(\pi_i, w_{k,u_i})]_{i=1}^{d_v}, \phi_k)$
    $\kappa_{k,v} \leftarrow h(r_{k,v})$
    if $\mathsf{Eligible}(v, e_k)$ and $v \notin D_k$ and G_p has an eligible stored record for v **then**
      $(o, A, t, \epsilon)_{k,v} \leftarrow \mathsf{VerifyIncumbentRecord}(G_p, v, \kappa_{k,v})$
    else if $\mathsf{Eligible}(v, e_k)$ and $\mathsf{ValidRecord}(\mathcal{M}, \kappa_{k,v})$ **then**
      $(o, A, t, \epsilon)_{k,v} \leftarrow \mathcal{M}[\kappa_{k,v}]$
    else
      $(o, A, t, \epsilon, \chi)_{k,v} \leftarrow \mathsf{Execute}(v, r_{k,v}, e_k)$
      if $\chi_{k,v}$ **then**
        $\mathsf{Store}(\mathcal{M}, \kappa_{k,v}, (o, A, t, \epsilon)_{k,v})$
      end if
    end if
    $w_{k,v} \leftarrow \mathsf{Payload}(v, o_{k,v}, A_{k,v}, \epsilon_{k,v})$
  end for
  $(y_k, \tau_k, c_k) \leftarrow \mathsf{Assemble}(G_k, \{(o, A, \epsilon)_{k,v}\}_{v \in V_k}, e_k)$
  $b_k \leftarrow \mathsf{HardFailures}(y_k, \tau_k)$
  $(q_k, \gamma_k) \leftarrow \mathsf{Judge}(x, G_k, y_k, \tau_k)$
  if judgment remains invalid after the permitted repair **then**
    $\mathcal{H}.\mathsf{Append}(\mathsf{Invalid}(k))$; **continue**
  end if
  $s_k \leftarrow \mathsf{RankingKey}(y_k, q_k, b_k, k)$
  $\mathcal{H}.\mathsf{Append}(k, G_k, y_k, \tau_k, q_k, \gamma_k, s_k, c_k)$
end for
if $\mathsf{Complete}(\mathcal{H}) = \varnothing$ **then**
  return failure
end if
return $y_{\arg\max^{\mathrm{prio}}_{i \in \mathsf{Complete}(\mathcal{H})} s_i}$

Here $p = \perp$ denotes a candidate synthesized without a parent, and $\epsilon_{k,v}$ is the execution-error record exposed to assembly and judging; an inherited record restores its stored ϵ, so a stored error still counts toward b_k. VerifyIncumbentRecord checks the incumbent record against $\kappa_{k,v}$, so a mismatch is a blocking integrity error rather than a silent miss. The flag $\chi_{k,v} \in \{0, 1\}$ indicates whether the result satisfies the node's eligibility contract and may enter $\mathcal{M}$. The write-capable sink is never eligible for either inheritance branch. A read-only node may contain a bounded multi-step tool loop. Its pre-execution request includes the rendered node instructions, ordered upstream payloads, effective execution limits, and execution-context fingerprint. Observations produced inside that loop are stored with the result and restored on an inheritance hit; they are not part of the pre-execution lookup key. $\mathsf{Complete}(\mathcal{H})$ is the set of indices of candidates with recorded execution and a valid judgment; Latest returns the most recent such index. $\mathcal{R}(G_p)$ lists the nodes whose individual removal leaves the workflow valid; Select is validated like Propose, so a discard set outside $\mathcal{R}(G_p)$ or a

kept workflow that fails validation is rejected, one JSON repair is permitted, and a second failure records an invalid proposal. $\mathsf{Restart}(\mathcal{H})$ holds when $\mathsf{Complete}(\mathcal{H})$ is empty, when no candidate in it has a valid output, or when $\max_{i \in \mathsf{Complete}(\mathcal{H})} q_i^{\mathrm{all}} \leq q_{i_0}^{\mathrm{all}} + 1$, where i_0 is the first completed candidate. ValidatedRestart supplies the highest-ranked candidate's judge feedback and workflow to the restart prompt (Appendix C.4). The restart is recorded as a full-graph transition with no workflow inheritance. It fires on 73/466 completed WorkBench transitions (15.7%) and 184/1,180 HotpotQA transitions (15.6%).

Throughout the paper, *initial workflow* means the first successfully executed candidate produced by initial synthesis. If synthesis fails before that point, the failed attempts remain in proposal-validity and cost accounting but are not treated as executable workflows.

C.2 Inherit-MAS Settings

Table 5 lists the settings shared by the reported Inherit-MAS runs on both benchmarks. Benchmark-specific roles, tools, output schemas, and runtime limits are given in Sections C.3 and C.6.

Table 5: Headline Inherit-MAS implementation settings shared across benchmarks unless noted.

Component	Frozen setting
Workers	GPT-4o-mini, temperature 0
Synthesis, node selection, refinement, judge	GPT-5.4-mini, separate judge prompt, temperature 0
Search	5 candidates, 1 Select decision and 1 validated edit per refinement, no early stop
Structured-output recovery	At most 1 JSON-repair call
Provider recovery	At most 1 identical retry for HTTP 500
HotpotQA retrieval	BM25, top 5, at most 4 calls/candidate
Inheritance scope	Task-local, request-exact, read-only nodes
Stateful commit	One write-capable sink, always executed live
API/runtime	Azure OpenAI-compatible endpoint

C.3 Workflow and Output Rules

Edit family and validation. The workflow specification declares each node's role, accepted inputs, produced output, and tool permissions; every edge must carry an allowed output to a compatible, ordered input. These explicit rules let the proposer choose from a finite menu of edits whose arguments can be checked before execution. An edit may change a node's prompt, assigned subtask, tool access, communication, or topology while still identifying where its influence begins. The controller is shared across benchmarks. Each adapter supplies the allowed node roles, input and output labels, input ordering, tools, output validator, and execution constraints through $\mathcal{I}$. The implemented edit family is

$$\begin{aligned}\mathcal{A}(G) \subseteq \{&\mathsf{Prompt}(v), \mathsf{Subtask}(v), \mathsf{Tools}(v),\\ &\mathsf{AddNode}(v), \mathsf{InsertNode}(u,v), \mathsf{DropNode}(v),\\ &\mathsf{AddEdge}(u,v), \mathsf{DropEdge}(u,v), \mathsf{Reorder}(v)\}.\end{aligned} \tag{8}$$

$\mathsf{InsertNode}(u,v)$ places a new node on an existing edge $u \to v$: the new node takes u's output and feeds v, and the edge $u \to v$ is retained. Only arguments permitted by these role, connection, and tool rules are instantiated in $\mathcal{A}(G)$. The validator enforces acyclicity, input/output compatibility and input cardinalities, identifier uniqueness, output reachability, authorized tools, and benchmark execution constraints.

HotpotQA. The synthesis payload contains this graph shape:

```
{
 "nodes": [
 {"id":"researcher_a", "role":"researcher",
 "subtask":"...", "system_prompt":"...",
 "tools":["search_fullwiki"], "inputs":[]},
 {"id":"answerer", "role":"synthesizer",
 "subtask":"...", "system_prompt":"...", "tools":[],
 "inputs":[{"from":"researcher_a","port":"evidence"}]}
 ],
 "output":"answerer"
}
```

A `planner` emits `plan` and has no inputs. A `researcher` emits `evidence` and may consume plans or evidence. A `verifier` emits `critique` and may consume evidence or critique. A `synthesizer` emits `answer` and may consume plans, evidence, or critique. Every edge port must equal its source role's emitted type. Only planners and researchers may be roots. The graph has two to seven nodes, exactly one synthesizer, at least one retrieval-capable researcher, and one synthesizer output that is the only sink.

WorkBench. The synthesis payload contains this graph shape:

```
{
 "name":"short name",
 "nodes":[
 {"id":"worker_a", "role":"worker", "subtask":"...",
 "system_prompt":"...", "tools":["declared read-only tool"],
 "inputs":[]},
 {"id":"integrator", "role":"integrator",
 "subtask":"combine evidence", "system_prompt":"...",
 "tools":[], "inputs":[{"port":"evidence_a","from":"worker_a"}]},
 {"id":"executor", "role":"executor",
 "subtask":"execute approved plan", "system_prompt":"",
 "tools":[], "inputs":[{"port":"plan","from":"integrator"}]}
 ],
 "sink":"executor",
 "rationale":"brief"
}
```

The allowed roles are `worker`, `critic`, `integrator`, `verifier`, and `executor`. Only workers may be roots or call read-only tools. Critics consume worker, critic, or integrator reports. Integrators consume worker or critic reports. A verifier receives exactly one plan from an integrator and may also receive worker or critic evidence. The graph contains at most eight language-model nodes and twelve ordered edges. Exactly one deterministic executor is the sole sink. It receives one plan from an integrator or verifier, has no prompt or tools, and is the only node allowed to execute state-changing calls.

C.4 Prompts, Scores, and Role Guides

This section reproduces the prompt templates used by the reported Inherit-MAS runs on both benchmarks. They are copied verbatim from the implementation. Line wrapping and quotation marks are typographic only. Task text, graph-specific instructions, upstream records, and tool schemas are substituted at runtime and serialized without paraphrasing. Every synthesized `subtask` and `system_prompt` is stored in the trajectory record.

Model roles. GPT-5.4-mini serves five controller roles: workflow synthesizer, node selector, edit proposer, structured judge, and one-shot JSON repair model. GPT-4o-mini executes every language-model node in the synthesized workflow. The WorkBench write executor and all schema validators are deterministic. HotpotQA retrieval uses the fixed BM25 service. Thus the judge and proposer use different system messages even though they use the same model family.

Initial workflow synthesis. The controller system message is:

> "Synthesize a small typed multi-agent workflow for the supplied task and interface. Use only the declared roles, ports, and tools. Decompose only when useful, keep information flow explicit, and end at the declared output node. Return JSON only."

Here, *typed* is the wording of the frozen prompt and means only that node roles, input/output labels, and tool permissions must follow the explicit workflow rules reproduced below. The user message is a JSON object with fields `benchmark`, `task`, `interface_contract`, and `workflow_contract`. The final-round restart appends "This is a stagnation escape; produce a meaningfully different valid workflow." Its payload adds `reason`, `prior_audit`, and `prior_graph`. We treat this restart as an implementation detail rather than a separate method component.

Structured judge. The judge system message is:

> "You are an independent, skeptical, gold-free evaluator of a task workflow. Derive the task's requirements from the public request and interface contract. Use only the proposed output, the observed artifacts in the audit, and the workflow's recorded node outputs. Penalize unsupported claims, missing requirements, invalid actions, and fluent guessing. Do not assume any benchmark-specific number of sources, steps, agents, or actions. When the node outputs show which workflow nodes contributed useful or useless work, name those node ids in the critique and in the keep/fix strings. Scores are 0–100. Return compact JSON with exactly the contract's fields and no others."

The user message contains `benchmark`, `task`, the public `interface_contract`, `proposed_output`, `artifact_audit`, the candidate's `workflow`, and its `node_outputs` (the compact execution trace), followed by this exact response contract:

```
{
 "quality_score": 0,
 "completion_likelihood": 0,
 "requirement_coverage": 0,
 "artifact_grounding": 0,
 "critique": "brief skeptical audit",
 "keep": ["component worth preserving"],
 "fix": ["highest-value repair"]
}
```

All four numeric fields must lie in $[0, 100]$. The critique must be nonempty. The validator retains at most four nonempty strings from each of `keep` and `fix`. Search cost is not part of `quality_score`. The online ranking first minimizes hard execution failures, then prefers a valid parsed output, `quality_score`, `requirement_coverage`, and `artifact_grounding`, in that order. The earlier candidate wins an exact tie.

The `artifact_audit` supplied to the judge has this common shape:

```
{
 "output_valid":true,
 "artifacts":[
 {"artifact_id":"...", "source":"...",
 "locator":"...", "observed":true,
 "content":"bounded observed content"}
 ],
 "execution_errors":{"node_id":"error"},
 "interface_diagnostics":{...}
}
```

For WorkBench, artifacts are read-only tool observations and interface diagnostics contain the proposed actions and invalid-output count. For HotpotQA, each artifact is a cited title and sentence index, and diagnostics contain declared and observed citation counts. The judge receives bounded observed content rather than a benchmark reference answer.

Node selection. The node-selection system message is:

> "Select which nodes of the latest workflow the next candidate inherits. Use the gold-free audit and execution diagnostics. Discard a node only when the audit or its recorded output shows that it contributes nothing useful or harms the result, and keep every other node. The kept workflow must remain valid under the workflow contract. Return compact JSON only."

Its JSON user message contains `benchmark`, `task`, `interface_contract`, `workflow_contract`, the incumbent `graph`, the judge `audit`, `artifact_audit`, a compact `execution` trace, `discardable_nodes`, and the following response contract:

```
{
 "discard": ["id of a node to drop; leave the list empty to keep every node"],
 "rationale": "why the kept nodes should be inherited"
}
```

`discardable_nodes` lists the nodes whose individual removal leaves the workflow valid, so the output node and any node that another node requires never appear in it. A discard set naming any other node, or one whose kept workflow fails validation, is rejected. When the list is empty, the call is skipped and the incumbent is inherited whole.

Edit proposal. The proposer system message is:

> "Choose exactly one available, prevalidated atomic workflow edit. Use the gold-free audit and execution diagnostics, preserve useful components, and address the highest-value reported defect. Do not emit a compound edit or copy a current value unchanged. Return compact JSON only."

Its JSON user message contains `benchmark`, `task`, `interface_contract`, the kept workflow as `graph`, the `selection` (discarded node ids and rationale), the judge `audit`, `artifact_audit`, a compact `execution` trace, an indexed `available_edits` menu, and the following response contract:

```
{
 "edit_index": 0,
 "rationale": "why this one atomic edit addresses the audit",
 "replacement": "only when required by the menu entry",
 "subtask": "only for an add-node edit that requires it",
 "system_prompt": "only for an add-node edit that requires it"
}
```

Previously tried indices for the same kept workflow are removed before prompting. Text edits also expose their current value and require a different replacement. The benchmark adapter enumerates only edits whose result passes graph validation. The WorkBench menu includes prompt and subtask replacement, worker-tool addition or removal, node removal, input reordering, edge addition or removal, and worker insertion on an existing edge. The HotpotQA menu includes prompt and subtask replacement, retrieval toggling, edge addition or removal, node removal, researcher addition, and researcher insertion on an existing edge.

Structured-output repair. The controller permits one JSON-repair attempt after a parsing or validation failure. Its system message is "Repair one JSON response. Address the exact validator error. Return JSON only." The JSON user message contains `validator_error`, `contract`, `invalid_response`, `original_system`, and `original_user`. A second failure remains an invalid proposal or judgment and is not silently resampled.

The compact execution trace contains aggregate live and inherited token equivalents, tool-call count, and one record per node. Each node record includes role, execution error, inheritance status, and a bounded output excerpt. WorkBench additionally includes active domains, parser failures, rejected artifact references, and the observed artifact count. HotpotQA includes the number of BM25 calls. This is the complete execution-diagnostic surface available to the edit proposer.

C.5 Worker Role Prompts

The following wrappers guide every GPT-4o-mini node. A synthesized node prompt is inserted at `NODE_INSTRUCTION`. Because this value is part of the graph, prompt edits change the resolved request and invalidate the corresponding stored record.

WorkBench shared wrapper. Every language-model node receives:

> "You are one read-only node in a multi-agent workflow. Follow your assigned subtask. Do not claim access to hidden gold outcomes. Do not execute state-changing actions.
> `NODE_INSTRUCTION`"

The role-specific suffix is one of the following:

- **Worker:** "Use assigned read-only tools when needed. Preserve concrete IDs, values, and complete findings in your final response."
- **Integrator:** "Produce only JSON in the following form. Use the fewest state-changing actions that fully satisfy the task. Exact schemas: `WRITE_TOOL_SCHEMAS`."
  `{"actions":[{"tool":"...","args":{...}}]}`
- **Verifier:** "Audit the proposed plan for task coverage, grounded arguments, and harmful extras. Return only JSON in the following form. `WRITE_TOOL_SCHEMAS`."
  ```
  {"verdict":"approve|revise",
   "actions":[...],"reasons":[...]}
  ```

- **Critic:** "Return a concise critique or reasoning artifact for downstream nodes."

The user-message template is:

```
CURRENT_DATETIME
Task: TASK_TEXT
Assigned subtask: NODE_SUBTASK
Upstream records: ORDERED_JSON_RECORDS
```

Each upstream record contains its input port, node identifier, role, text output, tool events, and error. The prompt expands `WRITE_TOOL_SCHEMAS` to the exact fourteen official WorkBench write signatures and their enumerated argument constraints. The synthesis interface also includes the exact upstream documentation for the thirteen read-only tools. These benchmark-owned schemas are inserted verbatim, including qualified tool names, parameter types, descriptions, and examples. The recorded source lock contains the complete expansion rather than a shortened rewording.

HotpotQA shared wrapper. Every node receives:

> "You are one node in a HotpotQA FullWiki multi-agent DAG. The retrieval tool is deterministic BM25. Use only retrieved observations and ordered upstream reports. Retrieved sentences are addressed by exact [title, sentence_id] pairs. Never invent a title, sentence index, or fact."

The wrapper appends exactly one role instruction:

- **Planner:** "Decompose the two-hop question into precise retrieval subgoals."
- **Researcher:** "Retrieve and report concise bridge/comparison evidence. Include exact [title, sentence_id] citations for every useful fact and distinguish evidence from hypotheses."
- **Verifier:** "Audit upstream evidence, resolve contradictions, and search only if necessary. Return the supported answer candidate and exact [title, sentence_id] citations."
- **Synthesizer:** "Return JSON only in the following form. Cite only observed indexed sentences."

```
{"answer":"short answer",
 "supporting_facts":[["exact title",0]],
 "confidence":0.0,"rationale":"brief"}
```

It then appends "Node instruction:" and the synthesized node system prompt. The user-message template is:

```
Question:
QUESTION_TEXT

Assigned subtask:
NODE_SUBTASK

Ordered upstream reports:
ORDERED_JSON_REPORTS
```

An upstream report contains its declared input port, source role, text output, and error.

C.6 Output Schemas and Runtime Limits

The WorkBench integrator returns an `actions` list whose entries contain a string `tool` and an object `args`. A verifier returns `verdict`, `actions`, and `reasons`. Only actions that pass the official tool-name and argument-schema validator reach the deterministic executor. Worker-role nodes run for at most eight turns, 1,024 output tokens per turn, and twenty read-only tool calls. Integrator, verifier, and critic calls have a 1,800-token output limit.

The HotpotQA synthesizer returns a string `answer`, a list of `supporting_facts`, numeric `confidence`, and string `rationale`. Every supporting fact is a title and integer sentence-index pair. A citation is valid only if the exact pair appeared in this candidate's retrieval observations. Each candidate receives four BM25 calls in total, allocated deterministically across retrieval-capable nodes, so adding or removing such a node changes the allowance of the others; a node whose allowance changes joins the affected region of Equation (3). Each call returns five documents. A tool-enabled node runs for at most five turns and all language-model nodes have a 900-token output limit.

For meta-model and judge calls, WorkBench uses output limits of 5,000 tokens for initial synthesis, 5,000 for refinement, and 3,000 for judging. HotpotQA uses 1,600, 900, and 500 tokens, respectively. Worker decoding uses temperature zero. The full resolved messages, model identity, decoding parameters, tool contract, execution-context fingerprint, and ordered inputs are recorded with every stored record.

C.7 Search Inputs and Offline Scoring

For WorkBench, the meta-model and judge receive the natural-language request, public tool names and schemas, tool observations produced by their own candidate, messages, parser errors, and timeouts. For HotpotQA, they receive the question and results of their own bounded BM25 calls. Reference actions, answers, supporting facts, and official metrics are reserved for post-run scoring.

The judge uses a system prompt separate from the synthesis/refinement prompt and additionally receives the candidate's workflow and each node's recorded output, which lets its critique name nodes. Its structured output contains four scores in $[0, 100]$: overall quality, completion likelihood, requirement coverage, and artifact grounding. The online ranking first rejects hard execution failures, then prefers valid output, overall quality, requirement coverage, artifact grounding, and finally the earlier candidate as a deterministic tie-break.

D Evaluation Protocol and Baselines

The WorkBench manifest contains 130 tasks, all tasks from 13 held-out template clusters. The HotpotQA manifest contains 300 public IDs selected by a deterministic hash and excludes all earlier development IDs. The WorkBench Overall column is the completion rate over all evaluation tasks, so domains contribute in proportion to their task counts: 20 each for Analytics, Calendar, Email, and Project management, 10 for CRM, and 40 for Multi-domain. HotpotQA joint F1 follows the benchmark evaluator: for each question, joint precision and recall are the products of the answer and supporting-fact precision and recall, joint F1 is their harmonic mean, and the table reports the mean over questions. Joint F1 therefore never exceeds the smaller of answer F1 and supporting-fact F1 and penalizes a correct answer without valid citations. Stored-record integrity audits pass for 1,859 inherited WorkBench results over 230 development tasks and 751 inherited HotpotQA results over 200 development questions.

D.1 External Baselines

The baseline implementations use pinned upstream releases with benchmark-specific adaptations. We retain released controllers where supported and otherwise adapt their search mechanisms to the benchmark interface. This subsection describes the GPT-4o-mini configurations; Appendix D.2 gives the Qwen configurations. The adaptations and their source/configuration provenance are documented below.

Model assignments and decoding. EvoAgent uses GPT-4o-mini across roles. EvoMAS and TacoMAS assign GPT-4o-mini to designated worker and compiler calls and GPT-5.4-mini to meta-model and judge calls, with EvoMAS's worker palette fixed to GPT-4o-mini. TacoMAS's additional answer-synthesis routing is specified in Appendix D.2. Native meta-model calls use temperature 0.7 for EvoMAS and 0.3 for TacoMAS; the HotpotQA TacoMAS adaptation uses temperature zero. Workers and judges use temperature zero, except that native EvoMAS workers follow the decoding settings in each evolved configuration.

EvoAgent. We port the population mechanism from commit `fc6d087` to the benchmark interfaces. The WorkBench configuration follows the released interactive path without a quality-filter call; HotpotQA retains role-quality filtering and permits four BM25 calls per agent. An auxiliary mixed-model configuration uses GPT-5.4-mini for role generation and quality filtering with GPT-4o-mini workers. It reaches 27.7% WorkBench completion and 30.2% HotpotQA joint F1, with 73.2% answer F1 and 38.4% supporting-fact F1. The HotpotQA comparison isolates model assignment. The WorkBench comparison also changes the quality-filter step, so its difference cannot be attributed to the model alone. We report the all-GPT-4o-mini configuration in the main tables.

WorkBench EvoMAS. We executed EvoMAS at commit `93fd9d6` as one serial dependent trajectory over the WorkBench manifest. The run retains cross-task memory, persistent per-domain pools, native mutation/crossover, model mutation, reward computation, experience consolidation, pool admission, parent rechecks, and fresh final execution. Its declared hyperparameters are matched to our search allowance: two parents, three dependent evolution rounds yielding five unique candidates, CodeAgent `max_steps=8`, and a 1,024-token per-request output cap. The model-mutation operator is retained, but the worker palette is fixed to GPT-4o-mini as on HotpotQA, and a post-run trace audit confirms that all 168.5M worker tokens executed on GPT-4o-mini. The fresh reevaluation of the selected winner completes 40 tasks (30.8%) using 173.59M provider tokens and $36.00. This is an algorithm-faithful hyperparameter match with matched role-level models, not a released-default reproduction.

HotpotQA EvoMAS. The GPT-4o-mini row uses the released controller at commit `93fd9d6` with a public task and output plugin. It retains serial task order, persistent pools and memory, two parents, two evolution steps per task, and model mutation within the GPT-4o-mini worker palette. Each native agent receives four calls to the pinned BM25 service. The controller executes independently of our graph representation and edit system.

The model-matched EvoMAS run completed one continuous state chain over all HotpotQA questions. A trace-level audit over 2,100 trajectory files and 10,535 worker traces found only GPT-4o-mini workers. A separate scoring amendment parses exact Python dictionary serialization without inferring content from prose. The source lock, trace audit, and sealed official score are retained in the experiment records.

HotpotQA TacoMAS. The main-table implementation adapts the fast capability and slow topology mechanisms from commit `6f0d545` to our shared execution runtime and structured answer/supporting-fact interface. It uses ten fast rounds, birth/death and rewiring checks every two rounds, five to twenty agents, at most two birth/death pairs and eight edge edits per update, and temperature zero. Each candidate shares four BM25 calls across its retrieval nodes, with the same sentence-mapping contract and no execution inheritance. This run uses GPT-4o-mini workers and GPT-5.4-mini meta-models and judges on the same manifest as the Inherit-MAS run. Qwen TacoMAS uses the same adaptation.

We also ran the released TacoMAS controller with a task and output plugin, preserving its native population, graph, ten fast rounds, and slow topology updates. It returned prose answers: 126 contained no citation and only 58 mentioned a sentence index. Its 33.2% answer F1 is retained as a released-interface diagnostic; missing structured supporting facts prevent an informative joint F1 comparison.

WorkBench TacoMAS. We executed TacoMAS at commit `6f0d545` on all main-table WorkBench tasks with its native AgentState, GraphManager, EvolutionController, five-agent initial population, birth/death, rewiring, ten fast rounds, and slow updates. Because the release does not implement WorkBench's canonical stateful executor, the benchmark bridge gives evolving agents thirteen canonical read-only tools and a non-mutating `done`. One post-evolution GPT-4o-mini compiler maps the final plan to official write schemas, and one deterministic executor applies the writes to a pristine sandbox. No TacoMAS state is projected into our graph representation. The run completes 45 tasks (34.6%) using 236.59M provider tokens and $50.72.

This is a benchmark adaptation of the released controller, not a verbatim upstream WorkBench implementation. We retain the older released-interface run, which solved 14/115 tasks, all of them no-action tasks, at 22.11M tokens and $12.10, only as an interoperability diagnostic. Unlike that run, the main-table adaptation is scored over the identical manifest with official execute-then-state-difference semantics.

Table 6: Descriptive released-interface TacoMAS diagnostic, retained separately from the main-table adaptation.

System	Tasks	Complete	Total live	USD
TacoMAS	115	14 (12.2%)	22.11M	12.10

The WorkBench EvoMAS run completes a continuous state chain over all WorkBench tasks, 390 dependent evolution rounds, 37,207 successful provider calls, 173.59M tokens, and $36.00 of estimated API spend. The model-matched HotpotQA EvoMAS run completes a continuous state chain over all HotpotQA questions, 600 evolutionary operations, 34,157 successful model calls, 112.28M tokens, and $29.74. Every observed worker trace in both runs uses GPT-4o-mini. The released-interface HotpotQA TacoMAS diagnostic run completes 2,992 fast rounds and 1,362 slow updates with 75,156 successful calls, 202.68M tokens, and $56.65. Two failed provider calls remain within otherwise completed trajectories. The WorkBench TacoMAS run completes 1,290 fast rounds and 578 slow updates with 39,699 successful native calls, 236.59M tokens, and $50.72. Its one oversized final plan is retained as an incorrect compiler outcome. All final outcomes remain in each official denominator.

D.2 Qwen3-32B Backbone Evaluation

The completed Qwen evaluation uses the same WorkBench and HotpotQA tasks across all five systems. Inherit-MAS replaces only its worker model, retaining its synthesis/refinement controller, judge, prompts, edit family, and scoring rules. Workers use Qwen3-32B in non-thinking mode at temperature zero, while synthesis/refinement and judging remain assigned to GPT-5.4-mini. EvoAgent, which has no separate meta-model, instead uses Qwen3-32B for every logical role, including role generation, integration, and HotpotQA role-quality filtering, through the same benchmark bridge as its GPT-4o-mini row and with no API calls. Serving uses vLLM 0.21.0 with bfloat16 weights, tensor parallelism two, a 32,768-token context, at most two concurrent sequences per replica, batch-invariant execution, CUDA graphs, and disabled automatic prefix caching. Model and tokenizer metadata and serving flags are recorded with the runs. Scheduling-only acceleration does not change these inference settings.

For WorkBench, the Qwen EvoMAS run uses its released evolution loop with two initial parents, three evolution rounds, five unique candidates, eight CodeAgent steps, and a 1,024-token worker completion limit. It preserves serial cross-task experience. The Qwen TacoMAS run retains the released fast/slow evolution mechanism and benchmark executor adapter. Task-solving calls, including worker-facing memory/top-k synthesis and final action compilation, are constrained to Qwen rather than allowed to fall back to the meta-model. In contrast, the historical GPT-4o-mini TacoMAS configuration retains native meta-model routing for memory/top-k answer synthesis. Thus, its cross-backbone comparison also includes this explicit role-routing correction. Native meta-model decoding settings are retained, including TacoMAS's temperature 0.3. These role constraints and per-method search settings are part of the reported implementation.

On HotpotQA, the Qwen EvoMAS and TacoMAS implementations share the execution runtime and structured answer/supporting-fact interface described in Appendix D.1. Qwen EvoMAS uses two parents, two evolution steps, four candidate attempts, and per-question population initialization without persistent cross-task experience; its GPT-4o-mini counterpart retains the released controller. Qwen TacoMAS uses the same graph adaptation as its GPT-4o-mini counterpart. Both Qwen adaptations use temperature zero, four BM25 calls per candidate, the same sentence-mapping contract, and no execution inheritance. Cross-backbone comparisons therefore include the stated implementation and model-routing differences.

Failed tasks remain in the denominator. Only completed, sealed runs with working retrieval infrastructure are scored. Local Qwen tokens and Azure meta-model and judge tokens are recorded separately. A local model's absence of an API charge does not imply zero inference cost, and raw token counts across different tokenizers are not treated as equal compute.

E Cost Accounting

E.1 Inference and Search Cost

In Table 7, *Worker* covers task-solving calls, including integration and benchmark-output compilation, while *Meta/judge* covers workflow synthesis, node selection, refinement, and critique. This is a role-based accounting split: EvoAgent uses GPT-4o-mini for its internal meta-model calls as well as its task-solving calls. Figure 5 reports the observed quality–cost positions, while Table 7 provides the provider ledger, and Tables 8 to 10 add inherited equivalents, wall time, and candidate-prefix

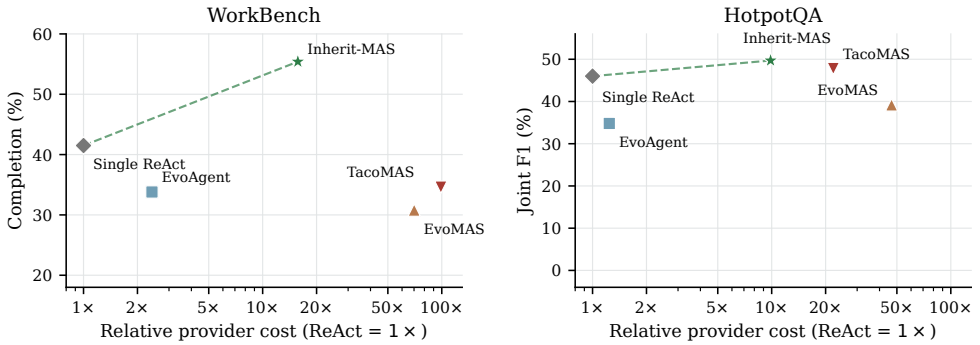

Figure 5: Observed quality–cost operating points on the complete benchmark manifests. Provider cost is normalized within each benchmark to Single ReAct $= 1\times$ using the configured deployment tariffs; the logarithmic horizontal axis favors lower cost and the vertical axis favors higher quality. The dashed segment joins the observed non-dominated points and is not an interpolation. Inherit-MAS is more accurate and less costly than EvoMAS and TacoMAS under these configurations, while Single ReAct remains a cheaper non-MAS reference.

costs. The role split exposes a different observed allocation of inference. Worker calls account for 13.9% of Inherit-MAS provider tokens on WorkBench, compared with 97.1% for EvoMAS and 94.0% for TacoMAS. On HotpotQA, the corresponding shares are 43.9%, 92.5%, and 51.8%. The absolute totals remain essential: EvoMAS and TacoMAS respectively use 16.8×/22.9× the Inherit-MAS WorkBench total and 13.7×/2.1× its HotpotQA total. Execution inheritance avoids 0.53M WorkBench and 1.93M HotpotQA worker-token equivalents, while meta-model and judge inference remains live. Under these configurations, Inherit-MAS combines stronger task-conditioned workflows, workflow inheritance, and execution inheritance to achieve a better observed quality–cost operating point than EvoMAS and TacoMAS. These provider-token counts are not price weighted, and the comparison is not a matched-budget intervention proving that one allocation is causally superior. Inherit-MAS is also not cheaper than Single ReAct because it performs task-specific search.

E.2 Detailed Cost Ledgers

The tables in this subsection give the per-run provider ledgers, the inherited token equivalents, wall time, and the candidate-prefix costs.

E.3 Inheritance Ledger and Full Traversal

This section completes the accounting of Section 4.4, using the inheritance indicator $z_{k,v}$ from Equation (5). The two structural cases of change localization have a direct interpretation. For $v \notin D_k$, the runtime verifies that the current request digest agrees with the incumbent node record before using its result. For $v \in D_k$, it constructs the request from the current upstream results and queries the same result store. An affected node can therefore still produce an inheritance hit when an affected path resolves to a request seen earlier. Otherwise, the node executes live and its eligible result is inserted as a new immutable record. An unchanged structure identifies the expected incumbent record, while the complete resolved-request digest remains the sole authority for execution inheritance.

If $t_{k,v}$ is the original worker-token usage associated with the stored or live result, the two ledgers are

$$\begin{aligned} T_{\text{live}} &= \sum_{k,v} (1 - z_{k,v}) t_{k,v}, \\ T_{\text{inherited}} &= \sum_{k,v} z_{k,v} t_{k,v}, \\ \rho &= \frac{T_{\text{live}}}{T_{\text{live}} + T_{\text{inherited}}}. \end{aligned} \tag{9}$$

Table 7: Live provider usage on the full evaluation sets. Token values are totals in millions, and inherited token equivalents are excluded. Relative provider cost normalizes each benchmark to Single ReAct $= 1.00\times$ using the configured deployment tariffs. On HotpotQA the Single ReAct harness also scores its one answer with the GPT-5.4-mini judge. Those calls do not change its output, and their cost is included in the reference, which raises the denominator of the HotpotQA relative-cost column.

System	Worker tokens	Meta/judge tokens	Total tokens	Relative cost
WorkBench				
Single ReAct (reference)	3.26	0	3.26	1.00×
EvoAgent	7.69	0.15	7.85	2.41×
EvoMAS	168.48	5.11	173.59	70.15×
TacoMAS	222.41	14.18	236.59	98.86×
Inherit-MAS	1.44	8.92	10.35	15.72×
HotpotQA				
Single ReAct (reference)	1.35	0.23	1.58	1.00×
EvoAgent	3.68	0.66	4.34	1.24×
EvoMAS	103.81	8.47	112.28	46.71×
TacoMAS	9.00	8.37	17.37	22.08×
Inherit-MAS	3.60	4.59	8.18	9.86×

Table 8: Full-manifest inference and search cost. Total live includes every provider token. Inherited equivalents are not added.

Benchmark	System	Total live	Inherited equiv.	Avg. live/task
WorkBench	Single ReAct	3.26M	0	25.1k
WorkBench	EvoAgent	7.85M	0	60.3k
WorkBench	EvoMAS	173.59M	0	1.335M
WorkBench	TacoMAS	236.59M	0	1.820M
WorkBench	Inherit-MAS	10.35M	0.53M	79.6k
HotpotQA	Single ReAct	1.58M	0	5.3k
HotpotQA	EvoAgent	4.34M	0	14.5k
HotpotQA	EvoMAS	112.28M	0	374.3k
HotpotQA	TacoMAS	17.37M	0	57.9k
HotpotQA	Inherit-MAS	8.18M	1.93M	27.3k

Here ρ is the fraction of the trajectory's worker-token volume executed live. An inheritance hit contributes zero live worker tokens. $T_{\text{inherited}}$ is a counterfactual token-equivalent and is never added to provider usage. A hit uses the previously observed result rather than drawing a new sample; the Inherit-MAS experiments therefore use temperature zero. Provider-side prompt caching is complementary because it still invokes the model, whereas execution inheritance removes the call.

Full execution executes every node. Full traversal with the same result store resolves every request digest before lookup. Inherit-MAS additionally uses the affected region to distinguish structurally unchanged nodes from request reconvergence inside that region, while still verifying a complete digest in both cases. Full traversal and the change-localized schedule therefore have the same live model-call miss set $\{(k, v) : z_{k,v} = 0\}$. We attribute worker-token savings to verified execution inheritance; the affected region provides change localization, auditability, and subgraph-level accounting rather than a separate source of worker-token savings.

E.4 Full-Execution Reruns and Store Footprint

The full-execution bars in Figure 4 come from independent reruns of the same Inherit-MAS controller with execution inheritance disabled, so every node executes live and the ledgers record actual provider usage. The two WorkBench reruns use 2.08M and 1.98M worker tokens and the HotpotQA rerun 5.50M, against 1.44M and 3.60M for the original runs. The reruns also check the inherited token equivalents recorded by the original runs. Charging every inheritance hit its stored original usage

Table 9: Separated Inherit-MAS accounting. Wall is summed task wall. Inherited equivalents are excluded from live columns.

Bench.	Worker	Meta/judge	Total	USD	Wall
WorkBench	1.44M	8.92M	10.35M	8.07	1.41h
HotpotQA	3.60M	4.59M	8.18M	6.28	3.34h

predicts 1.97M and 5.53M worker tokens under full execution, within 3% of the mean WorkBench rerun usage and the HotpotQA rerun usage, respectively. The deterministic-runtime audit below provides the evidence for execution equivalence under its tested conditions.

The persisted task-local result stores contain 979 WorkBench entries occupying 5.14 MiB (5.51 kB/entry) and 2,694 HotpotQA entries occupying 17.40 MiB (6.77 kB/entry). A post-hoc warm local-filesystem microbenchmark of record read, JSON decode, SHA-256, and in-memory rehydration takes median 0.080 ms (p95 0.122 ms) per WorkBench entry and 0.062 ms (p95 0.344 ms) per HotpotQA entry. These timings report storage-system scale only. They are not online latency measurements or evidence that the change-localized schedule outperforms full traversal with the same result store.

Deterministic inheritance qualification. Before the remote-API runs, we qualified the three execution conditions (full execution, full traversal with the result store, and the change-localized schedule) on a local deterministic Qwen3-14B and LiveCodeBench substrate. Across 240 frozen candidate-problem pairs, full execution, full traversal with the result store, and the change-localized schedule produced byte-identical node outputs and score vectors. The two inheritance conditions also had identical model-call sets and each used 17.7% of the full-execution worker-token volume. This audit validates request-verified execution inheritance on a qualified deterministic runtime. For the remote models, inheritance remains scoped to the stored request and output and does not assert fresh-call byte identity.

E.5 Candidate-Prefix Cost

Table 10 reports cumulative live cost as the test-time trajectory grows. Model wall sums worker execution and meta-model and judge call latencies and excludes deterministic orchestration. The fifth candidate is more expensive because the final-round restart frequently produces a new workflow rather than applying an inheritance-friendly local edit.

Table 10: Cumulative live cost through candidate prefixes. Tok. is total provider tokens. Model wall is summed model-call wall time.

Benchmark	Prefix	Tok.	USD	Model wall
WorkBench	1	2.20M	1.77	0.38h
	3	6.24M	4.87	0.86h
	5	10.35M	8.07	1.39h
HotpotQA	1	1.84M	1.22	0.86h
	3	4.97M	3.72	1.98h
	5	8.18M	6.28	3.30h

E.6 Artifacts and Accounting

Every run records a frozen manifest digest, source/configuration fingerprints, per-call provider usage, task-level wall time, tool or retrieval calls, invalid attempts, and atomic completion records. Candidate-generation, judge, worker, deterministic-tool, and benchmark-evaluator ledgers remain separate. Provider cache counters are recorded but are not treated as Inherit-MAS execution inheritance. A request-verified hit carries the original node usage as $T_{\text{inherited}}$ and records zero T_{live}. Stored record content and identity are digest-checked on every load. Any mismatch is a blocking integrity error.

F CODE AND DATA AVAILABILITY

The implementation, evaluation task lists, and scoring scripts are available anonymously at https://anonymous.4open.science/r/Inherit-MAS-ICLR2027-8202. WorkBench and HotpotQA data remain subject to their original licenses; restricted source data will not be redistributed.