TacoMAS: Test-Time Co-Evolution for Dynamic Multi-Agent Systems

Anonymous authors
Paper under double-blind review

Abstract

Multi-agent systems (MAS) have become a promising paradigm for tackling complex tasks. Recent studies have investigated self-evolving MAS that automatically improve agent capabilities and communication structures. However, existing approaches either learn a topology that remains fixed during inference or adapt only one of topology and capability at test time. We investigate jointly adapting both dimensions at different time scales: agent capabilities should evolve rapidly to respond to emerging subtasks, whereas topology should change more gradually to maintain coordination stability. Inspired by a stylized replicator-mutator model, we propose TacoMAS, a test-time co-evolution framework for dynamic MAS. TacoMAS casts MAS inference as online graph adaptation, in which nodes correspond to agents with role-specific capabilities and edges represent their communication topology. During inference, a fast capability loop updates agent expertise based on trajectory-level feedback, while a slower meta-LLM-driven topology loop evolves the MAS through agent birth-death operations, including edge editing, agent addition, and agent removal. Experiments across four benchmarks show that TacoMAS consistently outperforms nearly 20 multi-agent baselines, with an average improvement of 13.3 percentage points over the strongest baseline. Our code is publicly available at https://anonymous.4open.science/r/TacoMAS-3DD8/.

1 Introduction

LLM-based multi-agent systems (MAS) coordinate specialized agents for retrieval-intensive analysis and long-horizon planning (Wang et al., 2024; Guo et al., 2024; Händler, 2023). Their effectiveness depends on how agents are organized and responsibilities allocated, motivating automatic evolution of topology and capabilities (Li et al., 2024; Piccialli et al., 2025).

Previous work on evolving MAS can be broadly divided into training-time and test-time approaches. Training-time methods optimize the agent topology or role assignment once and keep it fixed during inference (Hong et al., 2023; Zhang et al., 2024; Shang et al., 2024; Wang et al., 2025; Hu et al., 2024). However, because the learned topology is fixed, it can easily mismatch unseen tasks whose latent subtasks and coordination demands deviate from the training distribution. Test-time methods instead treat inference as a dynamic evolution process, allowing MAS to adjust online based on intermediate states (Qian et al., 2024; Tastan et al., 2026; Qu et al., 2026). However, existing methods typically evolve either the communication topology (Qian et al., 2024; Tastan et al., 2026) or agent capabilities (Qu et al., 2026) alone. In fact, optimizing both aspects is essential; it is a key prerequisite for unlocking the full collaborative potential of MAS (Kim et al., 2025). This raises a question: how can we jointly adapt both topology and capabilities of MAS during inference?

Updating topology and capability at the same pace can destabilize coordination (Papoudakis & et.al, 2021): a verifier's improved checking strategy may become ineffective if simultaneous rewiring changes its evidence sources. This motivates fast capability refinement and slower structural adaptation (Fabiano et al., 2021; Mguni et al., 2023) (§3.5; §4.3).

Evolutionary game theory provides a useful interpretation: capability refinement reinforces effective behaviors under a fixed interaction graph, while slower structural adaptation changes that graph in response to accumulated feedback (Smith & Price, 1973; Taylor & Jonker, 1978; Borkar, 1997;

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.

Kushner & Yin, 2003; Nowak & Sigmund, 2004). We use this perspective only to motivate the update schedule: the surrogate model in §3.5 rests on assumptions that we state explicitly.

Motivated by this principle, we propose TacoMAS, which adapts both **T**opology and c**a**pability in a **co**-evolution framework for **MAS** during the inference of each query (Fig. 1). It consists of (1) a fast capability loop, where agents optimize their expertise based on their execution outcomes and contribution to the task in each round[1]; and (2) a slow meta-LLM-driven topology loop, which periodically reviews the trajectory and proposes a birth-death (BD) update with a small set of edge and agent edits. During the BD process, the meta-LLM decides which edges in the agent topology should be modified and whether to introduce a new agent or remove an ineffective one. The resulting team can revise its division of labor as the task unfolds.

Following the standard setup of recent multi-agent studies (Kim et al., 2025), we evaluate TacoMAS on four benchmarks spanning diverse task regimes: financial problem analysis, web browsing, Minecraft-style planning, and workplace task execution. Compared with nearly 20 MAS baselines, TacoMAS achieves an average improvement of 13.3 percentage points over the strongest baseline across the four datasets.

In summary, our key contributions are three-fold:

1. We highlight a key principle for test-time multi-agent evolution: agent capabilities and team topology should be adapted jointly, but on different time scales.
2. We propose TacoMAS, a test-time co-evolution framework that jointly adapts node capabilities and graph topology through two coupled loops. The design is inspired by a stylized replicator–mutator model whose assumptions we state explicitly (§3.5).
3. We conduct extensive experiments on four benchmarks. TacoMAS achieves the best performance on all datasets with an average improvement of 13.3 percentage points over the strongest baselines.

2 Related Work

Multi-agent LLM systems. Multi-agent systems extend tool-using and reflective agents (Yao et al., 2022; Shinn et al., 2023; Schick et al., 2023) with specialized roles for long-horizon and retrieval-heavy tasks (Zhou et al., 2023; Jimenez et al., 2023; Wei et al., 2025). The first generation of multi-agent frameworks coordinates a hand-crafted team of role-specialized agents: AutoGen (Wu et al., 2024) and MetaGPT (Hong et al., 2023) ship role templates and standardized operating procedures; CAMEL (Li et al., 2023) pairs a user agent with an assistant in a fixed dialogue loop; AgentVerse (Chen et al., 2023) and ChatDev (Qian et al., 2024) assembles role rosters per task category. *Limitation:* the graph and roster are designed once and held fixed; mid-instance signals cannot trigger new roles or rewiring.

Training- / Offline-evolving multi-agent systems. A second line replaces the human designer with automated search or learning, but the resulting artifact is still frozen at inference. Two families dominate. (i) Offline workflow/agent search produces one graph that all test queries share: AFlow (Zhang et al., 2024) explores workflow graphs with MCTS, AgentSquare (Shang et al., 2024) searches a modular "planning/reasoning/memory/tool-use" design space, ADAS (Hu et al., 2024) alternates a code-space designer with an executor, and EvoAgentX (Dang et al., 2025) mutates agent populations with evolutionary search. (ii) Trained per-query graph generators train a conditional generator once, then sample (and freeze) a fresh graph for each query: ARG-Designer (Li et al., 2026) autoregressively emits a DAG; MaAS (Zhang et al., 2025a) samples from a learned agentic supernet; MetaAgent (Zhang et al., 2025c) predicts an FSM of agent transitions; SwarmAgentic (Zhang et al., 2025b) assembles teams via a particle-swarm metaphor; MetaGen (Wang et al., 2026) and EvolveR-outer (Huang et al., 2026) likewise regenerate the roster / routing per query with only constrained execution-time edits. These designs primarily pay the organization cost before task execution, limiting their response to evidence revealed during a trajectory.

Test-time evolving multi-agent systems. A growing line updates the MAS during an instance, treating "inference time" as a dynamic process. Existing methods, however, each commit to a single

[1]Capability refinement here updates contextual memory and instructions, not model weights.

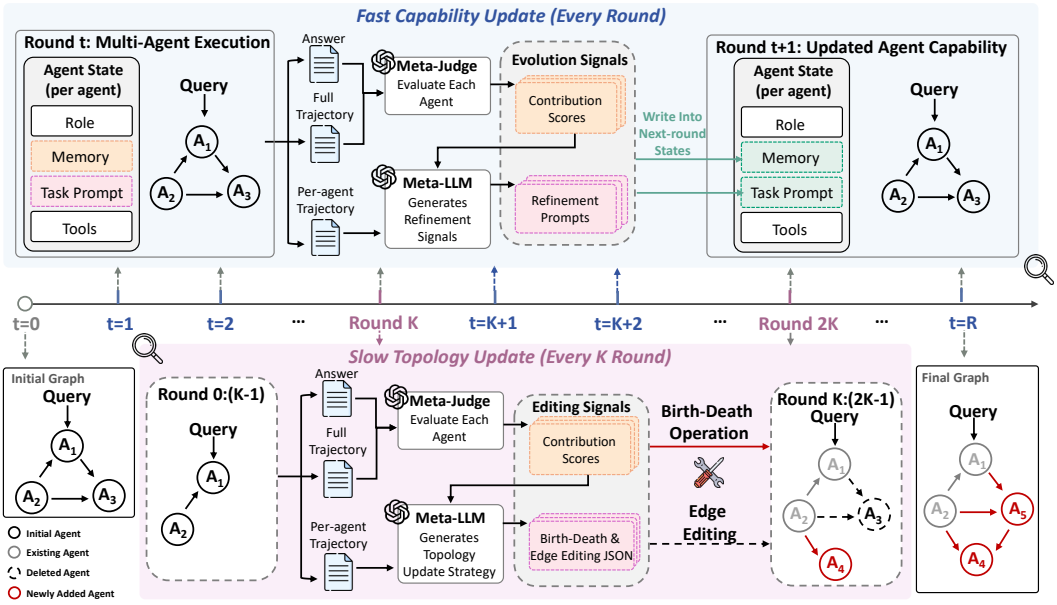


Figure 1: The overview framework of TacoMAS.

update axis. (i) Topology-only: ChatDev-Puppeteer (Dang et al., 2025) has a centralized orchestrator pick the next persona over a fixed pool; SelfOrg (Tastan et al., 2026) rebuilds a top-k communication DAG every round from response-similarity Shapley scores. In both, agent prompts and tool policies are fixed. (ii) Capability-only: CORAL (Qu et al., 2026) updates a shared memory and skill bank in a long-running loop, while the topology stays implicit. TacoMAS couples these two axes: capability refinement operates within the current graph, while bounded structural updates respond to persistent coordination failures. Evolutionary-game and two-time-scale models provide a conditional analytical perspective on this design (Taylor & Jonker, 1978; Nowak & Sigmund, 2004; Hofbauer & Sigmund, 1998; Akin, 2013; Borkar, 1997; Kushner & Yin, 2003).

3 Method: TacoMAS

The overview of our proposed framework is illustrated in Figure 1; Algorithm 1 gives the complete procedure.

3.1 Setup and Notation

Multi-agent system setting. Given a query q, one MAS execution round produces an answer a through the workflow defined by agent roles, capabilities, and communication.

Test-time evolution. Unlike static systems, we evolve the MAS online during the inference of each query. We formalize the system as a directed agent graph $\mathcal{G}_t = (T_t, \Phi_t)$ indexed by execution round $t = 0, 1, \ldots, R$, which decouples into a topology $T_t = (\mathcal{V}_t, \mathcal{E}_t)$, with agents $\mathcal{V}_t$ and directed information channels $\mathcal{E}_t \subseteq \mathcal{V}_t \times \mathcal{V}_t$, and a capability state $\Phi_t = \{\phi_{v,t}\}_{v \in \mathcal{V}_t}$, where $\phi_{v,t}$ comprises an agent's prompt, contextual memory, and tool inventory. A Meta-LLM $\mathcal{M}$ initializes $\mathcal{G}_0$ and orchestrates its evolution; agents instantiate roles from a fixed pool $\mathcal{R}$ (*e.g.,* Planner, Searcher, Verifier).

3.2 Two-time-scale Dynamics

The central design of TacoMAS is the asynchronous co-evolution of agent capabilities Φ and topology T on two distinct time scales. This joint update process is formulated as:

$$\Phi_{t+1} = F^C(\Phi_t; T_t, \xi_t), \qquad T_{t+K} = F^T(T_t; \Phi_{t:t+K}, \xi_{t:t+K}), \tag{1}$$

Algorithm 1 TacoMAS Procedure

Input: query q; round cap R; slow interval K; edit budget $\mathbf{B} = (B_{\mathcal{V}}, B_{\mathbb{E}})$; threshold τ.
$\mathcal{G}_0 \leftarrow \mathcal{M}_{\text{init}}(q)$; $t \leftarrow 0$.
repeat
  for $v \in \mathcal{V}_t$ **do** ▷ fast capability loop F^C
    execute tool policy of v under $(\phi_{v,t}, \mathbb{E}_t)$, observe $r_{v,t}$
    $c_{v,t} \leftarrow \mathcal{J}(r_{v,t})$; $\phi_{v,t+1} \leftarrow F^C(\phi_{v,t}; r_{v,t}, c_{v,t})$ equation 2
  read answer a_t, score $s_t \leftarrow \mathcal{J}(a_t, q)$
  if $(t+1) \bmod K = 0$ **and** $s_t < \tau$ **then** ▷ slow topology loop F^T
    $\Delta T, \{\Delta\phi_v\} \leftarrow \mathcal{M}_{\text{slow}}(T_t, \Phi_{t-K:t}, \{c_{v,\tau}\})$
    enforce $|\Delta\mathcal{V}| \le B_{\mathcal{V}}$, $|\Delta\mathbb{E}| \le B_{\mathbb{E}}$; $T_{t+1} \leftarrow T_t \oplus \Delta T$ equation 3
  $t \leftarrow t+1$
until $s_t \ge \tau$ **or** $t \ge R$ **or** STOP
return a_t from the sink agent of $\mathcal{G}_t$.

where F^C and F^T denote capability and topology operators, and K is the slow-update interval. Every execution round, agents incorporate trajectory feedback into their reasoning and tool-use strategies under the current graph. The slow operator uses accumulated feedback to reconfigure that graph. Separating the updates gives recent capability refinements time to operate before their information sources or collaborators change.

3.3 Fast Capability Loop F^C

Within each execution round t, agents act under fixed topology T_t using role instructions and contextual memory. Agent v produces a trajectory $r_{v,t}$ of reasoning, tool outcomes, and messages; together these form $\xi_t = \{r_{v,t}\}_{v \in \mathcal{V}_t}$.

Capability update via memory refinement. The meta-judge assesses each agent's contribution from ξ_t, returning $c_{v,t}$ and a justification. The meta-LLM combines this feedback with $r_{v,t}$ to diagnose errors and propose an execution plan. These corrections enter the agent's contextual prompt in the next round, updating $\phi_{v,t}$ through memory refinement. The evaluation and refinement prompts appear in App. E.

Capability-update abstraction. To reason about this process we model capability evolution as a discrete replicator-style update (Hofbauer & Sigmund, 1998) over abstract behavioral weights $w_v > 0$, a "selection pressure" that reallocates influence toward higher-contributing behaviors:

$$w_{v,t+1} \propto w_{v,t} \exp\big(\eta(c_{v,t} - \bar{c}_t)\big), \quad \bar{c}_t = \tfrac{1}{|\mathcal{V}_t|} \sum_{u \in \mathcal{V}_t} c_{u,t}, \tag{2}$$

where $\eta > 0$ controls the update strength. This is a population-level abstraction of the textual refinement, not a literal parameter update. One possible empirical link is the following conditional drift hypothesis, which we state but do not verify.

Assumption 1 (Replicator-bias of the meta-LLM update)**.** There exist $\eta > 0$ and slack $\epsilon_{\text{noise}} \ge 0$ such that, for every fast round t, $\mathbb{E}[\bar{c}_{t+1} \mid \mathcal{H}_t] \ge \bar{c}_t + \eta\, \mathbb{V}\text{ar}_v[c_{v,t}] - \eta\, \epsilon_{\text{noise}}$, where $\mathcal{H}_t$ is the history of trajectories and scores up to round t.

It implies neither a gradient representation nor task-accuracy improvement; the aggregate diagnostics in App. C.3 are descriptive, not verification.

3.4 Slow Topology Loop F^T

While the fast capability loop optimizes per-agent capability, the slow update F^T reconfigures the MAS topology by modifying the sets of agents $\mathcal{V}$ and edges $\mathcal{E}$. After every K rounds, the meta-LLM $\mathcal{M}$ proposes a structural delta $\Delta T = (\Delta\mathcal{V}, \Delta\mathcal{E})$ to resolve systemic bottlenecks that individual capability refinement cannot fix.

Per-agent birth-death and edge edits. The structural delta ΔT is realized through two operations: 1) Birth-Death: A *birth* introduces a new agent role to expand functional capacity, while a *death* removes agents whose contribution scores c_v remain consistently low, a discrete mutation of the population support. 2) Edge Reconfiguration: $\mathcal{M}$ adds or removes communication channels to repair information flow, *e.g.,* a new edge from a high-contribution searcher to a verifier that lacks context. Both operations are implemented via textual prompts (App. E).

Update stability. To maintain the stability of the two-time-scale dynamics, we introduce edit budgets on the structural update:

$$T_{t+K} = T_t \oplus \Delta T, \quad |\Delta\mathcal{V}| \le B_{\mathcal{V}}, \ \ |\Delta\mathcal{E}| \le B_{\mathcal{E}}, \tag{3}$$

where $B_{\mathcal{V}}$ and $B_{\mathcal{E}}$ represent the maximum allowed edits for agents and edges, respectively. These budgets limit structural volatility to reduce disruption of recently refined capabilities.

Initialization and termination. The meta-LLM $\mathcal{M}$ seeds the initial graph $\mathcal{G}_0$ by selecting roles from the pool $\mathcal{R}$; we set $|\mathcal{V}_0| = 5$. The evolution process terminates when one of the following conditions is met: 1) the global score s_t, produced by an execution-time LLM judge with no access to the ground-truth rubric (§4), reaches the threshold τ; 2) the execution reaches the maximum round budget R; or 3) the meta-LLM issues a stop signal upon detecting convergence in the agent trajectories.

3.5 A Surrogate Model that Inspired the Design

The fast–slow schedule was inspired by a replicator–mutator picture: capability updates reinforce useful behaviors under the current graph, while bounded topology edits change roles and communication paths. We make this picture concrete through an idealized surrogate process. Everything in this section is an *assumption* about that surrogate, not a property of LLM agents: its state is a normalized behavioral weight vector $\pi_v = w_v / \sum_u w_u$ over agents, updated by the normalized form of Eq. equation 2, and a topology T, whereas real capability states are text (§3.3). Proofs and a fast-step drift bound appear in App. A.

Assumptions. Let $f_v(\pi, T)$ be a bounded reference contribution, $|f_v| \le F$, and $U(\pi, T) = \sum_v \pi_v f_v(\pi, T)$. We assume U is C^2 with uniformly bounded derivatives and $\partial_v U = f_v + \lambda(\pi, T)$ with λ common to all coordinates (potential compatibility), and that the surrogate observes $c_v = f_v + \zeta_v$ with $|\zeta_v| \le \epsilon_0$ almost surely; write $\epsilon = 2F\epsilon_0$. Fix a nonempty target topology set $\mathcal{T}_\star$ chosen independently of the current topology, let $d(T)$ be its graph-edit distance, and let $G(\pi, T) = \max_{\pi'} U(\pi', T) - U(\pi, T) \in [0, M]$ be the capability gap.

Assumption 2 (Conditional topology drift)**.** For each slow update, conditioned on the complete pre-edit history, $\mathbb{E}[d(T^+) \mid \mathcal{H}^-] \le (1-\gamma) d(T^-) + \epsilon_{\text{meta}}$ with $0 < \gamma < 1$, and $\mathbb{E}[G(\pi^+, T^+) \mid \mathcal{H}^-] \le G(\pi^-, T^-) + C_B$, where $C_B = C(B_{\mathcal{V}} + B_{\mathcal{E}})$.

None of these conditions is automatic, and none has been verified for LLM agents.

Theorem 1 (Expected surrogate bound per slow cycle)**.** *Under the assumptions above, let $L = d + \eta G$. A cycle of K fast updates followed by one slow update obeys $\mathbb{E}[L_{n+1} \mid \mathcal{H}_n] \le (1-\gamma) L_n + \widetilde{\epsilon}$ with $\widetilde{\epsilon} = \epsilon_{\text{meta}} + K\eta^2\epsilon + \eta C_B + \gamma\eta M + C_f K \eta^3$ for a uniform constant C_f; hence $\mathbb{E}L_N \le (1-\gamma)^N L_0 + \widetilde{\epsilon}/\gamma$.*

The bound couples the two time scales: fast steps contribute only second-order noise $K\eta^2\epsilon$, whereas each slow edit pays a budget-dependent price ηC_B, so bounded, infrequent structural edits are compatible with contraction. This intuition led to TacoMAS, nothing more: the bound concerns the surrogate potential, not task accuracy, may be vacuous, and does not select K or the budget. Whether the design helps is settled empirically (§4; grading variability in App. D.6).

4 Experiments

We evaluate overall performance (Q1), backbone robustness (Q2), update settings (Q3), the roles of capability and organization (Q4), and a concrete evolution trajectory (Q5).

Table 1: Main accuracy results, grouped by evolution time scale. Best accuracy per dataset in bold, best baseline underlined; the last row gives the absolute gain (percentage points) of TacoMAS over the best baseline.

Method	finance	browsecomp	plancraft	workbench
Fixed-topology (Kim et al., 2025)				
SAS	<u>0.539</u>	0.200	0.530	0.347
MAS-Independent	0.529	0.090	0.710	0.416
MAS-Decentralized	0.445	0.240	0.600	0.446
MAS-Centralized	0.500	0.270	0.560	0.386
MAS-Hybrid	0.537	0.260	0.540	0.386
Offline-evolved				
MetaGPT (Hong et al., 2023)	0.348	0.550	0.668	0.473
AFlow (Zhang et al., 2024)	0.343	0.633	0.717	0.478
AgentSquare (Shang et al., 2024)	0.286	0.625	0.574	0.440
EvoAgentX (Wang et al., 2025)	0.313	0.591	0.763	0.419
ADAS (Hu et al., 2024)	0.294	0.595	0.497	0.645
Per-instance graph design				
AgentVerse (Chen et al., 2023)	0.301	0.625	0.653	0.523
ARG-Designer (Li et al., 2026)	0.295	0.621	0.619	0.595
MaAS (Zhang et al., 2025a)	0.284	0.620	0.492	0.538
MetaAgent (Zhang et al., 2025c)	0.349	0.601	0.560	0.469
SwarmAgentic (Zhang et al., 2025b)	0.338	0.570	<u>0.812</u>	<u>0.651</u>
MetaGen (Wang et al., 2026)	0.419	0.513	0.479	0.492
EvolveRouter (Huang et al., 2026)	0.332	0.170	0.320	0.290
Within-instance evolution				
ChatDev-Puppeteer (Qian et al., 2024)	0.340	0.603	0.553	0.441
SelfOrg (Tastan et al., 2026)	0.377	<u>0.688</u>	0.712	0.441
CORAL (Qu et al., 2026)	0.409	0.505	0.458	0.511
Topology-capability co-evolution				
TacoMAS	**0.767**	**0.745**	**0.887**	**0.824**
Gain (p.p.)	**+22.8**	**+5.7**	**+7.5**	**+17.3**

Experimental setup. Following Kim et al. (2025), we evaluate on four benchmarks: `finance` (Bigeard et al., 2025) (reasoning over SEC filings), `browsecomp-plus` (Chen et al., 2025) (multi-hop retrieval and entity disambiguation), `plancraft` (Dagan et al., 2024) (crafting and feasibility planning), and `workbench` (Styles et al., 2024) (tool-based workplace workflows). Accuracy is the mean per-instance fraction of rubric items passed (correctness and contradiction criteria). All task agents use Gemini-2.5-flash-lite unless otherwise stated; the default rubric judge is GPT-4o-mini and the TacoMAS controller is Gemini-2.5-pro. Final accuracy uses each benchmark's ground-truth rubric; the round score s_t used at test time for termination and controller feedback comes from a separate GPT-4o-mini judge call with a self-generated rubric, so no component sees the ground-truth rubric or reference answer during inference. Default settings are $R = 10$, $K = 2$, $|\mathcal{V}|_{\max} = 20$, at most two birth/death pairs and four edge edits per slow update, starting from a centralized five-role graph; the labeled interval sweep instead uses budget $(2, 8)$.

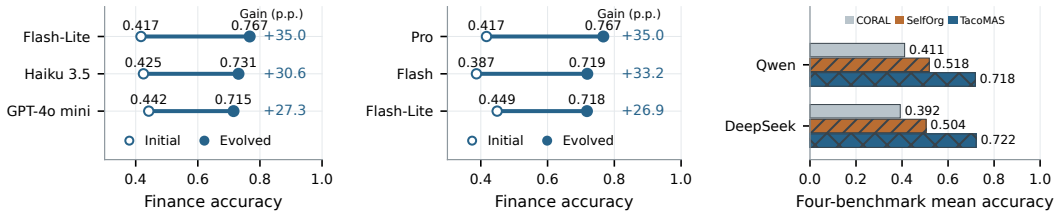
4.1 Q1: Overall Performance

Comparison protocol. We compare with 20 baselines spanning fixed topology, offline evolution, per-instance graph design, and within-instance evolution (App. C.1). Methods share the task-agent backend and dataset-specific tools unless stated otherwise. Each uses its own execution protocol; tokens and dollar costs are not matched (App. D.5).

Results across task regimes. TacoMAS leads on all four benchmarks (Table 1). The strongest comparator varies: single-agent execution on Finance, self-organization on BrowseComp, and per-instance graph design on planning and workplace tasks. Gains are largest on Finance and WorkBench and remain positive on retrieval and planning, including over within-instance methods based on

routing, rewiring, or memory adaptation. These results motivate the coupled design but do not isolate its contribution from added compute; Q3 and Q4 examine settings and mechanisms.

4.2 Q2: Robustness to Model Backbones



(a) Task-agent backbone (b) Meta-controller backbone (c) Open-weight backbones

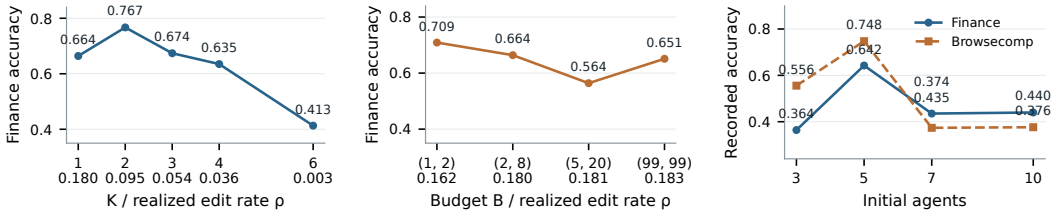
Figure 2: **Robustness across backbones.** Initial and evolved Finance accuracy for task agents and controllers, followed by four-benchmark mean accuracy against CORAL and SelfOrg under each open-weight family.

Task-agent backbones. Evolution improves Finance accuracy with Gemini, Claude, and GPT agents (Fig. 2(a)); each backend benefits relative to its own initialization.

Meta-LLM backbones. Pro gives the best final result, but Flash and Flash-Lite controllers retain substantial gains (Fig. 2(b)). Even using Flash-Lite for both controller and task agents improves on initialization. A stronger controller helps, but is not required for improvement; Q4 examines whether it alone explains the advantage.

Open-weight backbones. Qwen3.5 pairs a thinking `122b-a10b` controller with non-thinking `35b-a3b` task agents; DeepSeek-v4 pairs `pro` with `flash`. TacoMAS outperforms CORAL and SelfOrg on all four benchmarks under both families (Fig. 2(c) shows the means; App. D.3 gives per-benchmark results).

4.3 Q3: Ablations of the Co-Evolution Design



(a) Interval K, $B = (2, 8)$ (b) Budget B, $K = 1$ (c) Initial team size

Figure 3: **Update timing, edit allowance, and team size.** The first two panels show complementary slices of the interval/budget sweep, with realized edit rate ρ under each setting. The sweep is not factorial. Initial-team-size results use their archived scoring summary and available completed runs.

Update interval and edit budget. We vary the slow-update interval K and budget $B = (B_{\mathcal{V}}, B_{\mathcal{E}})$ in complementary slices (Fig. 3(a),(b)). The realized structural edit rate is

$$\rho = \frac{b}{K\,\overline{|\mathcal{V}| + |\mathcal{E}|}}, \tag{4}$$

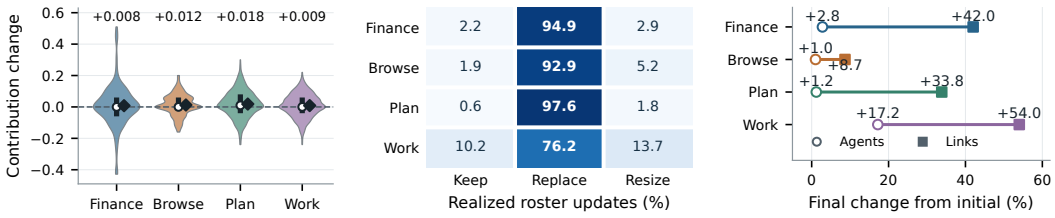
where b counts edits actually applied per slow update, rather than the permitted maximum. At fixed budget, shorter intervals substantially raise ρ, but accuracy peaks at an intermediate interval. At fixed interval, a larger budget changes ρ only modestly and does not improve accuracy: the controller leaves much of the extra allowance unused.

An intermediate adaptation regime. Frozen topology and forced rewriting, the two stress endpoints of the sweep, both underperform intermediate adaptation. The best setting, $K = 2$, $B = (2, 8)$,

reaches 0.767, compared with 0.709 for $K = 1$, $B = (1, 2)$. The sweep's frozen arm and Q4's fixed-graph arm were run under different protocols; we report both as observed, not as comparable. The sweep is not factorial and lacks repeated-run uncertainty, so it establishes neither a universal optimal rate nor an effect due solely to ρ. Full results appear in App. D.1; Q4 compares the original capability/topology update schedules.

Initial team size. Both retrieval benchmarks favor an intermediate initial size (Fig. 3(c)): larger teams spend more calls without consistently improving accuracy. These descriptive, unpaired comparisons suggest that team size is a coordination choice rather than a reliable scaling knob.

4.4 Q4: Why Does Co-Evolution Help?



(a) Contribution changes (b) Member updates (c) Agent and link changes

Figure 4: **Capability refinement and team reorganization.** (a) Round-to-round change in the meta-judge's mean contribution score (bars, dots, diamonds: interquartile range, median, mean). (b) Executed membership updates among valid reviews (API-error fallbacks excluded); replacement keeps team size. (c) Agent and link changes from first to last recorded step. Longer trajectories contribute more pairs in (a) and reviews in (b).

4.4.1 Capability Refinement and Team Reorganization

Does feedback improve agent execution? The fast loop uses the meta-judge's contribution scores to revise each agent's instructions for the next round. Round-to-round changes in the team's mean contribution score have mildly positive means but substantial zero and negative mass (Fig. 4(a)). Refinement is uneven, and because pairs come from continuing trajectories that may straddle topology updates, this diagnostic does not isolate a fast-loop treatment effect. Q5 illustrates how feedback requests source-grounded evidence from an existing searcher.

Does adaptation do more than add agents? Replacing members while keeping the team size unchanged dominates membership updates (Fig. 4(b)), while WorkBench shows more resizing. The slow loop thus changes participation with limited population growth. Replacement may preserve role names and does not by itself establish better specialization.

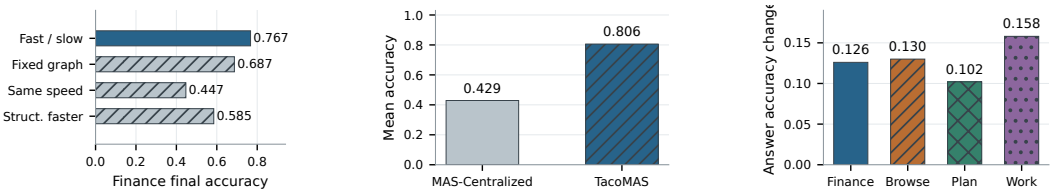
Does communication structure matter? Links grow more than the population on every benchmark (Fig. 4(c)), so adaptation changes information routes beyond enlarging the team. These executed edits establish reorganization but do not isolate the accuracy benefit of communication; the schedule ablation below tests the combined update.

4.4.2 Performance of Joint Adaptation

Does separating the updates help? The original Finance ablation favors fast capability refinement with slow topology updates over fixed structure, same-speed updates, and faster structural updates (Fig. 5(a)). This supports organizational repair with time for refinement between repairs, without establishing a factorial interaction or isolating each edit operation under matched compute.

Does a stronger meta-LLM explain the gains? A stronger controller improves final accuracy, but the all-Flash-Lite setting in Q2 still benefits from evolution, and TacoMAS outperforms MAS-Centralized when both use a Pro controller with Flash-Lite agents (Fig. 5(b)). Controller strength alone thus does not explain the advantage, though compute is not matched.

Does further evolution improve answer quality? Each round's answer is scored by the execution-time judge (no ground-truth access). Figure 5(c) compares rounds 1 and 6 using the score at each

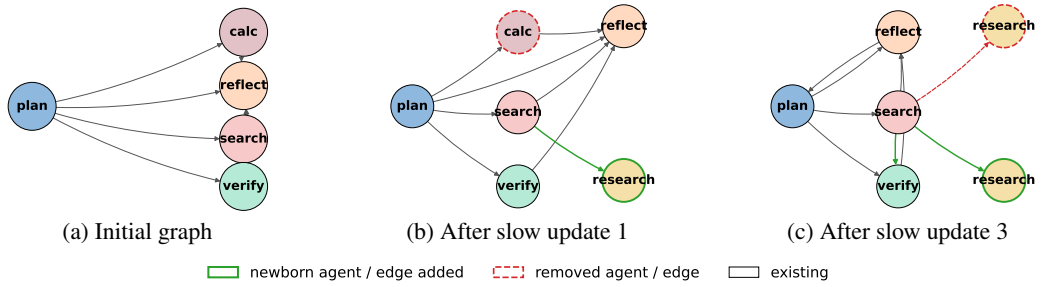


(a) Schedule ablation (b) Same-controller baseline (c) Answer improvement

Figure 5: **Performance of joint adaptation.** (a) Finance schedule ablation, original protocol; its fixed-graph arm differs from the sweep's frozen arm (App. D.1). (b) Four-benchmark mean with the same Pro controller and Flash-Lite agents; compute not matched. (c) Round-6 minus round-1 answer accuracy under the execution-time judge (self-generated rubric, no ground truth), not the final evaluation.

round rather than the best so far. Accuracy improves on all four benchmarks, suggesting that continued adaptation benefits answers as well as contribution scores, although intermediate rounds can regress, as on Finance. These are execution-time scores, not the final evaluation in (a,b), and later rounds use more computation, so this is not an equal-compute advantage (App. D.2).

4.5 Q5: Case Study of Task-Conditioned Evolution



(a) Initial graph (b) After slow update 1 (c) After slow update 3

Figure 6: The original representative Finance case: initial team, first slow update, and third slow update. Role replacement and communication changes specialize the team for retrieval and evidence checking.

Figure 6 follows the original Finance case about U.S. Steel's planned merger with Nippon Steel. The initial team produces overlapping summaries with insufficient document-level evidence. The controller replaces the calculator with a researcher and directs search output toward that role; when the bottleneck persists, later updates replace the researcher again, request more targeted extraction, and revise routes. Capability feedback requests source-grounded evidence, while structural updates change the members and routes for producing it. Later rounds still contain unsupported summaries: coordinated repair, not immediate success. Further traces appear in Apps. C.2, C.8, and C.9.

5 Conclusion

TacoMAS jointly refines agent capabilities and communication topology within each task via fast feedback-driven refinement and slower bounded structural updates. It achieves the best accuracy on all evaluated benchmarks, and ablations and trajectory analyses support separating the two updates. More broadly, TacoMAS suggests that multi-agent systems need not rely on fixed, hand-designed structures: treating capabilities and topology as co-evolving components offers a principled and practical path toward self-organizing multi-agent systems. We discuss limitations and future directions in App. B.

AI USE STATEMENT

Generative AI tools assisted experimental work and literature search. They also assisted the revision of this manuscript, including checking consistency, plotting figures, and helping to prove a certain theorem. The authors take responsibility for the final text, research claims, experimental results, references, and other artifacts, including material prepared with AI assistance.

REFERENCES

Ethan Akin. *The geometry of population genetics*. Springer Science & Business Media, 2013.

Antoine Bigeard, Langston Nashold, Rayan Krishnan, and Shirley Wu. Finance agent benchmark: Benchmarking llms on real-world financial research tasks. *arXiv preprint arXiv:2508.00828*, 2025.

Vivek S Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5): 291–294, 1997.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, and et.al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2023.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, and et.al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: an evaluation dataset for planning with llm agents. *arXiv preprint arXiv:2412.21033*, 2024.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, and et.al. Multi-agent collaboration via evolving orchestration. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Francesco Fabiano, Marianna Bergamaschi Ganapini, and et.al. Epistemic planning in a fast and slow setting. In *TFSOCTAI@ AAAI Fall Symposium*, 2021.

Taicheng Guo, Xiuying Chen, Yaqi Wang, and et.al. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Thorsten Händler. Balancing autonomy and alignment: A multi-dimensional taxonomy for autonomous llm-powered multi-agent architectures. *arXiv preprint arXiv:2310.03659*, 2023.

Josef Hofbauer and Karl Sigmund. *Evolutionary games and population dynamics*. Cambridge university press, 1998.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, and et.al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Jiatan Huang, Zheyuan Zhang, Kaiwen Shi, Yanfang Ye, and Chuxu Zhang. Evolverouter: Co-evolving routing and prompt for multi-agent question answering. *arXiv preprint arXiv:2604.05149*, 2026.

Carlos E Jimenez, John Yang, Alexander Wettig, and et.al. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, and et.al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Harold J Kushner and G George Yin. *Stochastic approximation and recursive algorithms and applications*. Springer, 2003.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36, 2023.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, 2026.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1, 2024.

David Henry Mguni, Haojun Chen, and et.al. Mansa: learning fast and slow in multi-agent systems. In *International Conference on Machine Learning*, 2023.

Martin A Nowak and Karl Sigmund. Evolutionary dynamics of biological games. *science*, 303(5659): 793–799, 2004.

Georgios Papoudakis and et.al. Agent modelling under partial observability for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Francesco Piccialli, Diletta Chiaro, and et.al. Agentai: A comprehensive survey on autonomous agents in distributed ai for industry 4.0. *Expert Systems with Applications*, 291, 2025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, and et.al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, 2024.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, and et.al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, and et.al. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36, 2023.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36, 2023.

J Maynard Smith and George R Price. The logic of animal conflict. *Nature*, 246(5427):15–18, 1973.

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. *arXiv preprint arXiv:2405.00823*, 2024.

Nurbek Tastan, Samuel Horváth, and Karthik Nandakumar. Stochastic self-organization in multi-agent systems. In *The Fourteenth International Conference on Learning Representations*, 2026.

Peter D Taylor and Leo B Jonker. Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1-2):145–156, 1978.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, and et.al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18, 2024.

Yimeng Wang, Jiaxing Zhao, Hongbin Xie, and et.al. Metagen: Self-evolving roles and topologies for multi-agent llm reasoning. *arXiv preprint arXiv:2601.19290*, 2026.

Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. Evoagentx: An automated framework for evolving agentic workflows. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2025.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, and et.al. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, and et.al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, and et.al. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, and et.al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

Yao Zhang, Chenyang Lin, Shijie Tang, Haokun Chen, Shijie Zhou, Yunpu Ma, and Volker Tresp. Swarmagentic: Towards fully automated agentic system generation via swarm intelligence. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025b.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines. *arXiv preprint arXiv:2507.22606*, 2025c.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, and et.al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Proofs and Limits of the Surrogate Analysis

A.1 Population-Frequency View

For positive abstract weights w_v (§3.3), let $\pi_v = w_v / \sum_u w_u$; the surrogate update is the normalized form of Eq. equation 2, $\pi_v^+ = \pi_v \exp(\eta c_v) / \sum_u \pi_u \exp(\eta c_u)$. This is an analytical model of behavioral reinforcement, not an operation on prompt strings or trained model weights. Compact simplices and the uniform regularity assumptions suffice; a finite capability state space is neither assumed nor implied by a bounded number of agents.

A.2 Fast-Step Derivation

Proposition 2 (Conditional fast-step drift)**.** *Under potential compatibility and the bounded score error of §3.5, the normalized update equation 2 with sufficiently small $\eta > 0$ satisfies*

$$\mathbb{E}[U(\pi_{t+1}, T) \mid \mathcal{H}_t] \geq U(\pi_t, T) + \eta \operatorname{Var}_{\pi_t}(f) - \eta\epsilon - C_f \eta^2, \tag{5}$$

for a uniform finite constant C_f, where $\mathcal{H}_t$ is the history up to round t. Positive drift is ensured only when $\operatorname{Var}_{\pi_t}(f) > \epsilon + C_f \eta$; the drift concerns the surrogate potential, not task accuracy.

Proof of Proposition 2. Uniform boundedness of c gives $\pi_v^+ - \pi_v = \eta \pi_v (c_v - \langle c \rangle_\pi) + O(\eta^2)$, with a uniform remainder in ℓ_1. Taylor expansion of U and potential compatibility cancel the common λ term, yielding

$$U(\pi^+, T) - U(\pi, T) = \eta \operatorname{Var}_\pi(f) + \eta \sum_v \pi_v (f_v - \bar{f}) \zeta_v + R, \qquad |R| \leq C_f \eta^2.$$

Since $|f_v - \bar{f}| \leq 2F$, the noise term is bounded below by $-2\eta F \epsilon_0 = -\eta\epsilon$. Taking conditional expectations proves the stated inequality, retaining the discretization term. □

For the corresponding continuous replicator flow, potential compatibility gives $\dot{U} = \operatorname{Var}_\pi(f)$. Without that condition this identity is false in general. For example, for two types with frequency p, take $f_1 = 1 - p$, $f_2 = 0$. Then $U = p(1-p)$ and $\dot{p} = p(1-p)^2$, so $\dot{U} < 0$ for $p > 1/2$ despite positive fitness variance. A scalar observed mean-improvement trend cannot establish the missing potential condition.

A.3 Cycle Bound

Proof of Theorem 1. During the fast phase T is fixed. Dropping the nonnegative variance term and iterating the conditional bound yields

$$\mathbb{E}[G^- \mid \mathcal{H}_n] \leq G_n + K\eta\epsilon + C_f K \eta^2, \qquad \mathbb{E}[L^- \mid \mathcal{H}_n] \leq L_n + K\eta^2\epsilon + C_f K \eta^3.$$

The slow-phase assumptions imply

$$\begin{aligned} \mathbb{E}[L^+ \mid \mathcal{H}^-] &\leq (1-\gamma) d^- + \epsilon_{\text{meta}} + \eta G^- + \eta C_B \\ &\leq (1-\gamma) L^- + \epsilon_{\text{meta}} + \eta C_B + \gamma\eta M. \end{aligned}$$

The tower property, and $1 - \gamma \leq 1$, give the claimed cycle bound. Iteration sums a geometric series:

$$\mathbb{E} L_N \leq (1-\gamma)^N L_0 + \frac{1 - (1-\gamma)^N}{\gamma} \widetilde{\epsilon} \leq (1-\gamma)^N L_0 + \widetilde{\epsilon}/\gamma.$$

The index counts slow cycles, not individual execution rounds. □

The target set must be specified independently. Taking all local optima of every topology would make the projected topology distance identically zero whenever each topology admits an optimum. Likewise, a probability of increasing the topology's fitness ceiling does not imply decreasing graph distance. Our explicit drift assumption avoids these inferences, but substantially limits the theorem's applicability. Observational noise measurements in App. D.6 characterize the available data; they cannot certify the conditions of this theorem.

B Limitations and Future Direction

A single meta-controller can become a coordination bottleneck on long, decomposable tasks, limiting scalability. Future work could explore hierarchical or distributed multi-controller architectures that combine global topology adaptation with local task coordination, enabling parallel local decisions while maintaining global workflow consistency.

Moreover, resetting scratch memory between task instances preserves instance independence but limits the reuse of structural discoveries. Future work could investigate cross-instance structural memory that stores effective topologies, role assignments, and collaboration patterns from previous tasks as retrievable experience, allowing selective transfer based on the requirements of new tasks. Such mechanisms would require reliable filtering and isolation to prevent negative transfer from irrelevant experience and leakage of task-specific information.

C Additional Experimental Details and Results

C.1 Baseline Details

We compare with 20 baselines grouped by adaptation time scale. All methods use the same dataset-specific tools and base LLM unless stated otherwise.

Fixed-topology (Kim et al., 2025). SAS uses one tool-augmented agent. MAS-Independent runs multiple agents independently and aggregates by voting. MAS-Decentralized uses a complete communication graph. MAS-Centralized uses a leader-worker structure. MAS-Hybrid uses a leader with worker sub-clusters.

Offline-evolved. These methods optimize workflow or agent design before deployment and freeze it at inference: MetaGPT (Hong et al., 2023), AFlow (Zhang et al., 2024), AgentSquare (Shang et al., 2024), EvoAgentx (Wang et al., 2025), and ADAS (Hu et al., 2024).

Per-instance graph design. These methods generate or select a graph for each query, then keep it fixed during execution: AgentVerse (Chen et al., 2023), ARG-Designer (Li et al., 2026), MaAS (Zhang et al., 2025a), MetaAgent (Zhang et al., 2025c), SwarmAgentic (Zhang et al., 2025b), MetaGen (Wang et al., 2026), and EvolveRouter (Huang et al., 2026).

Within-instance evolution. These methods adapt during inference but only along one axis. ChatDev-Puppeteer (Qian et al., 2024) selects among fixed personas. SelfOrg (Tastan et al., 2026) rewires a communication DAG while keeping prompts fixed. CORAL (Qu et al., 2026) updates memory and skills while leaving topology implicit. TacoMAS jointly evolves both topology and capabilities.

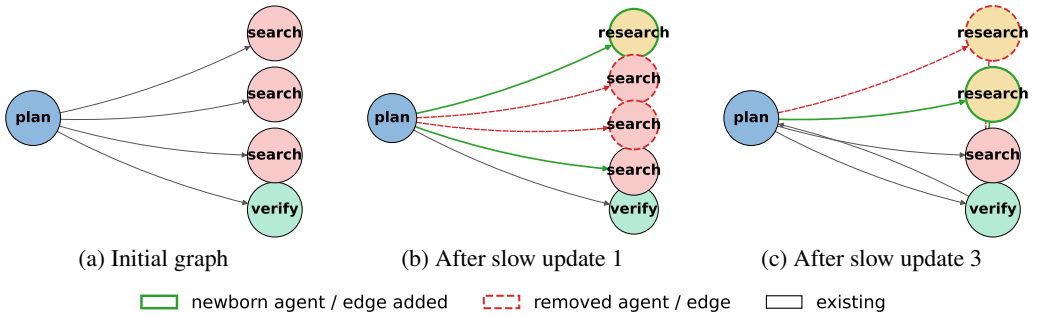
C.2 Evolution Traces on Other Datasets

Figures 7–9 show representative evolution traces on the remaining datasets. Each trace visualizes the system state at three stages: the initial centralized graph, the graph after the first slow update, and the graph after three slow updates or convergence. These snapshots illustrate how the topology is progressively reorganized as the fast-phase feedback accumulates. Green edges indicate newly added connections, while red dashed edges indicate removed connections, making the structural changes between successive stages explicit.

C.3 Assumption Verification

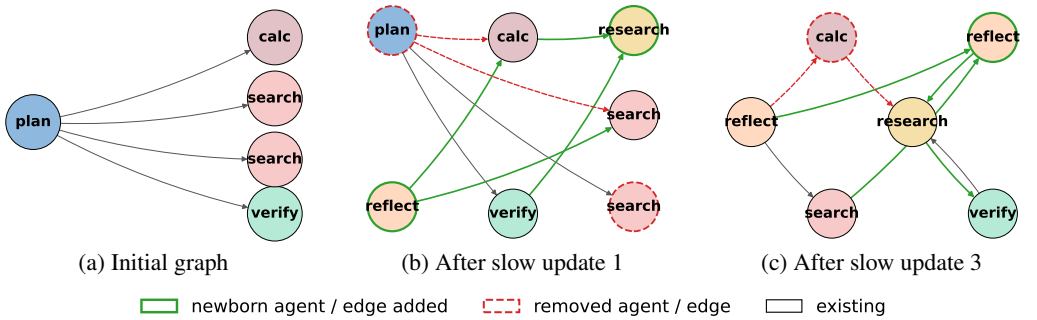
We examine contribution drift as a descriptive check of the directional bias in Assumption 1. Positive mean change is compatible with beneficial refinement, but an unconditional aggregate does not verify the history-conditional assumption. Continuing trajectories can also differ from those that stop early.

Result. The mean contribution increment is positive on all four benchmarks, although many individual increments are non-positive. Positive average drift can therefore coexist with frequent



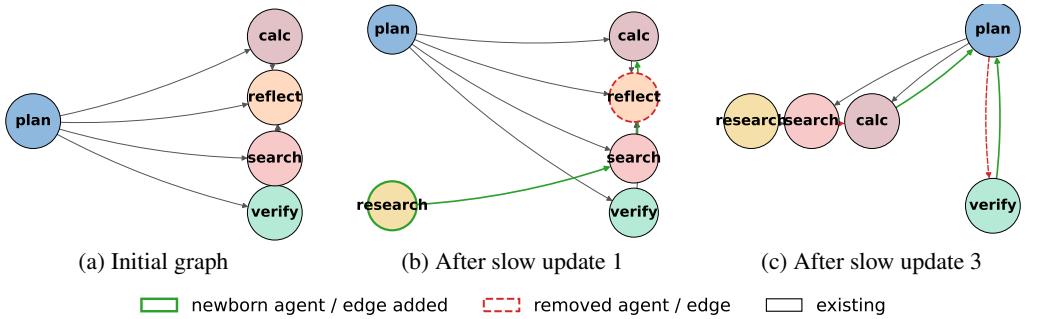
(a) Initial graph (b) After slow update 1 (c) After slow update 3

Figure 7: `browsecomp-plus` trace.



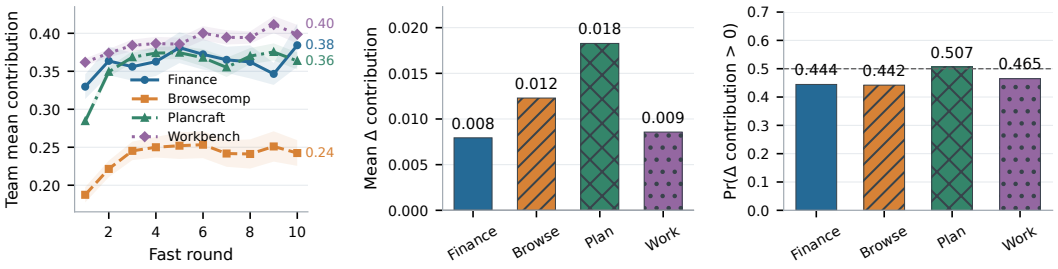
(a) Initial graph (b) After slow update 1 (c) After slow update 3

Figure 8: `plancraft` trace.



(a) Initial graph (b) After slow update 1 (c) After slow update 3

Figure 9: `workbench` trace.



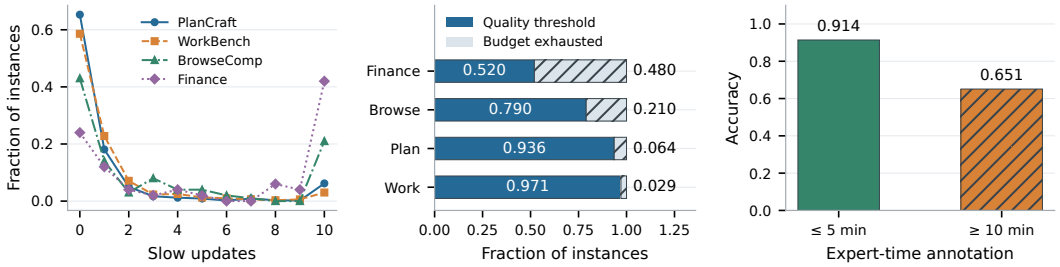
(a) Mean contribution by round (b) Mean contribution change (c) Fraction of positive changes

Figure 10: Contribution statistics across fast rounds. Left: team mean contribution (shading: SEM over available trajectories). Middle: mean round-to-round change. Right: fraction of positive changes. Late-round means condition on trajectories that continue; positive aggregate drift is not a conditional or causal guarantee.

negative or zero changes. This is consistent with the directional bias motivating the analysis, but does not establish the cause of negative increments or verify every condition of the theorem. Additional variance and structural-edit checks are in App. D.6.

C.4 Slow-Update Counts

Figure 11 shows how structural adaptation is distributed across tasks. Planning and workplace trajectories often terminate early; retrieval tasks have a longer budget-limited tail. This describes demand under the chosen execution cap, without establishing that extending the cap would improve unresolved answers.



(a) Slow-update distribution (b) Stopping behavior (c) Finance difficulty

Figure 11: Adaptive execution: slow-update distributions, stop reasons, and Finance accuracy by expert-time tier. Expert-time annotations are not visible to the model.

C.5 Fine-Grained Breakdown

Table 2 reports per-subcategory accuracy and median slow-update count. Subcategories come from dataset metadata: expert time for `finance`, question type for `workbench`, answer pattern for `plancraft`, and expected-answer length for `browsecomp-plus`. The relationship depends on the benchmark: expert-time and answer-length tiers separate evolution demand, whereas tool-count categories alone do not consistently predict more slow updates.

Table 2: Per-subcategory accuracy and median slow-update count.

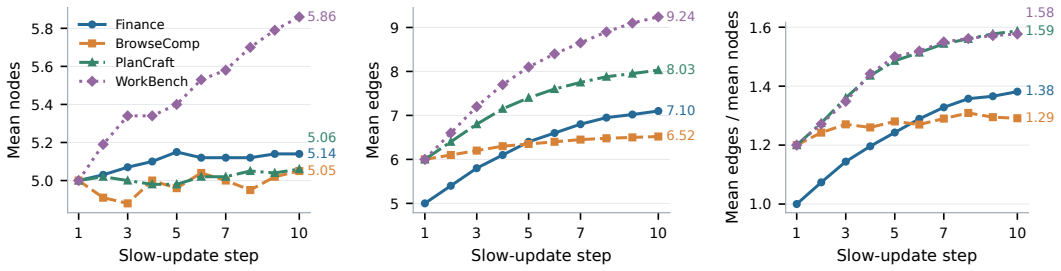
Dataset	Subcategory	mean Acc	median slow upd.
`finance`	easy (`expert_mins` ≤ 5)	0.914	1
`finance`	hard (`expert_mins` ≥ 10)	0.651	10
`plancraft`	feasible	0.874	0
`plancraft`	impossible	0.940	0
`workbench`	single: analytics	0.838	1
`workbench`	single: calendar	0.764	0
`workbench`	single: email	0.792	0
`workbench`	single: crm	0.681	1
`workbench`	single: project_mgmt	0.825	0
`workbench`	two-tool composite	0.907	0
`workbench`	multi-tool (≥ 3)	0.933	0
`browsecomp-pl.`	short answer (≤ 10 chr)	0.803	0
`browsecomp-pl.`	medium answer (11–25)	0.733	2
`browsecomp-pl.`	long answer (> 25)	0.647	4

C.6 Stop Reasons

Figure 11 reports whether instances stop by reaching the answer-quality threshold or by exhausting the round budget. Budget exhaustion records unresolved execution under a fixed cap; it does not establish that further evolution would help. These diagnostics describe resource allocation, while the main text emphasizes contribution refinement and structural change.

C.7 Graph Densification

TacoMAS edits both nodes and edges. Figure 12 shows that node count stays relatively stable, while edge count grows consistently. Thus, the slow loop mainly rewires existing agents rather than spawning many new ones, supporting the bounded-edit design.



(a) Agent count (b) Communication edges (c) Edges per agent

Figure 12: Graph evolution: average node count, average edge count, and the ratio of these averages. Edge growth exceeds node growth in the reported trajectories.

C.8 Case Study: FINANCE Instance 17

In this additional Finance case, the archived score improves at the final round after repeated retrieval specialization. A retrieved-source identifier and judge acceptance do not independently authenticate a primary filing. Table 3 distinguishes realized graph changes from the controller's intentions.

Table 3: Realized replacements and selected graph changes on FIN0017. The proposed first-step edge re-typing is not present in the stored graph snapshots.

Step	Rationale	Birth/death	Edge edits
1	Retrieval bottleneck; calculator irrelevant.	− calculator; + researcher	reroute searcher→new researcher
2	Searcher stagnates; evidence not reaching verifier.	− searcher; + researcher#2	add researcher→verifier
3	Researchers cite secondary sources.	− researcher#2; + primary-filing searcher	add planner→reflector
4	Need stricter 10-K/10-Q retrieval.	− searcher'; + 10-K/10-Q searcher	inherit predecessor routes

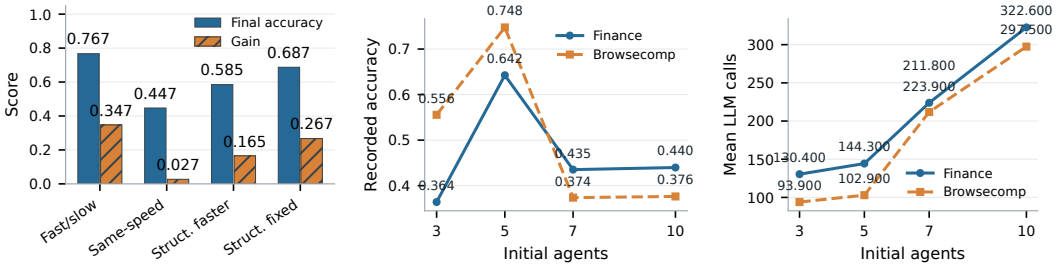
This case illustrates topology-capability coupling: the controller repeatedly diagnoses the same retrieval bottleneck but escalates specialization rather than applying unrelated edits.

C.9 Contrast Case: FINANCE Instance 8

We also include an easier finance case asking how many basis points MU beat or missed its Q3 2024 GAAP gross-margin guidance. The initial graph fails to retrieve the guidance number, but a single slow update removes the irrelevant calculator and adds a researcher specialized in extracting named figures from filings. The next round retrieves the relevant actual and guidance margins, computes the difference, and stops. This contrast shows that TacoMAS spends evolution budget only where the instance requires it: some tasks need repeated specialization, while others are solved after one targeted structural edit.

C.10 Original Schedule and Backbone Analyses

Figure 13 retains the original schedule and agent-count analyses. The original schedule categories and the extended K sweep use different experimental settings and should not be treated as interchangeable arms. For the initial-agent-count experiment, both retrieval benchmarks peak at five agents; completed-instance counts vary across settings. This sweep was not extended to PlanCraft or WorkBench.



(a) Original update schedules (b) Initial team size (c) Cost of larger teams

Figure 13: Original schedule comparison, initial team size, and call cost. Agent-count points use the available completed runs; no missing scores are imputed.

Table 4: Initial-agent-count sweep from its archived evaluation summary. Completed samples differ across arms; these are descriptive, unpaired comparisons.

Dataset	Initial $\|V\|$	Accuracy	Calls
Browsecomp	3	0.556	93.9
Browsecomp	5	0.748	102.9
Browsecomp	7	0.374	211.8
Browsecomp	10	0.376	297.5
Finance	3	0.364	130.4
Finance	5	0.642	144.3
Finance	7	0.435	223.9
Finance	10	0.440	322.6

C.11 Archived Finance Inference Trajectory

Figure 14 preserves the original Finance trajectory. The archived loader carries early-stopped instances forward and combines best-so-far online scores with stored terminal scores. The horizontal axis counts agent iterations. Because the endpoint and intermediate scores use different evaluation conventions, this is a descriptive diagnostic, not a uniform-metric scaling law.

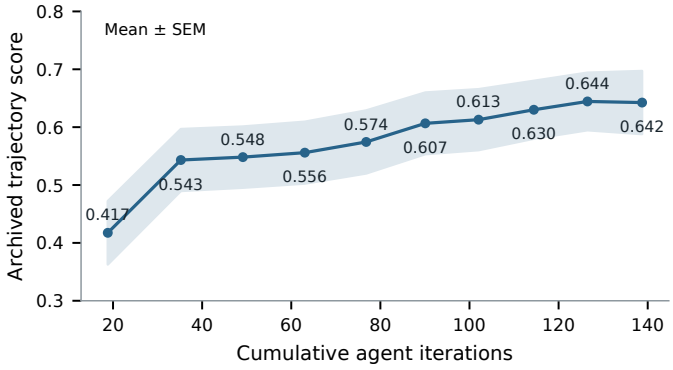


Figure 14: Archived Finance inference trajectory; shading denotes SEM. The aggregation mixes online and terminal scoring conventions.

D Supplementary Analyses and Robustness

D.1 Structural Update-Rate and Budget Sweep

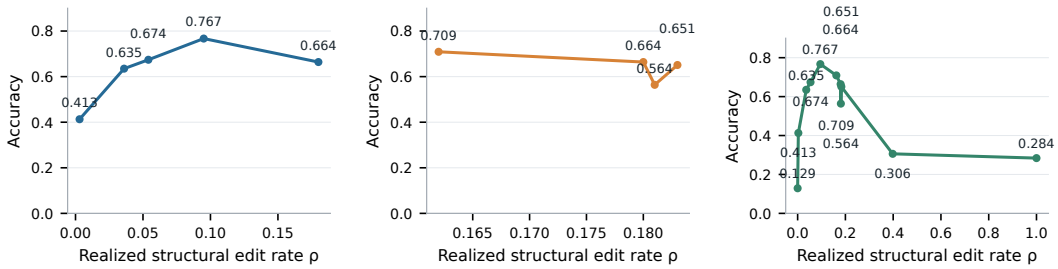
Table 5 reports the consolidated same-protocol sweep. Intermediate structural change performs best; frozen topology and forced rewriting underperform. Relaxing the nominal budget induces only

modest additional editing, because the controller need not use every available operation. At $K = 1$, budget $(1, 2)$ reaches 0.709, compared with 0.767 at $K = 2$, budget $(2, 8)$. Thus, useful performance is not confined to one setting, although sensitivity remains. The reported WorkBench interval sweep is less sensitive; individual arms and repeated-run uncertainty were not supplied. The frozen arm of this sweep (0.129) and the fixed-graph arm of the original schedule ablation (0.687, Fig. 5a) were run under different protocols (App. C.10). We report both numbers exactly as obtained; they are not two estimates of the same quantity and should not be compared with each other.

Equation 4 assumes a slow update every K fast rounds. With early stopping or skipped updates, an empirical rate can instead be measured as

$$\widehat{\rho} = \frac{\text{total applied structural edits}}{\text{total fast rounds } |\mathcal{V}| + |\mathcal{E}|}.$$

This separates the configured interval from observed execution. When counting edits, budgets and measurements must use the same unit: a birth/death pair is distinct from two individual node operations.



(a) Interval sweep (b) Budget at K = 1 (c) Full sweep by realized rate

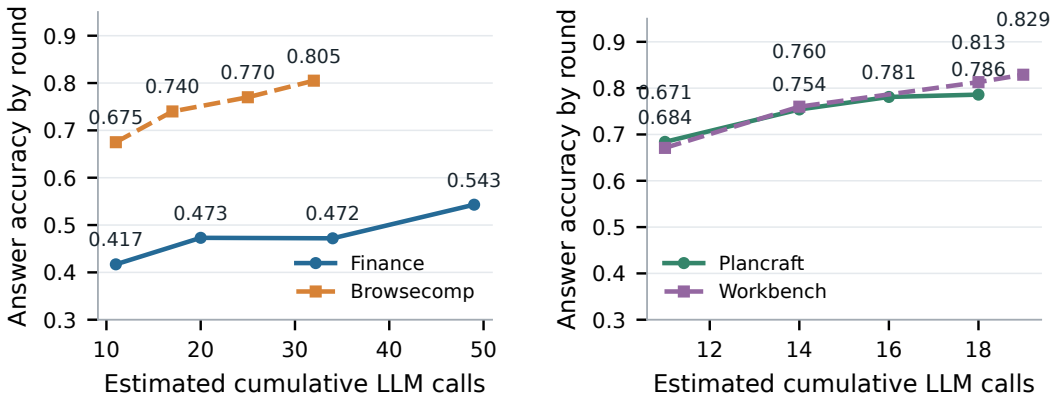
Figure 15: Three views of the reported structural sweep. Lines connect the displayed settings and are guides to the eye; the experiment does not isolate structural rate from all other factors.

Table 5: Consolidated Finance structural sweep from the updated rebuttal. These settings are reported under the same protocol; varying the realized rate also changes interval, budget, or forced-edit policy.

Setting	ρ	Accuracy
frozen, B=(0,0)	0.000	0.129
K=6, B=(2,8)	0.003	0.413
K=4, B=(2,8)	0.036	0.635
K=3, B=(2,8)	0.054	0.674
K=2, B=(2,8)	0.095	0.767
K=1, B=(1,2)	0.162	0.709
K=1, B=(2,8)	0.180	0.664
K=1, B=(5,20)	0.181	0.564
K=1, B=(99,99)	0.183	0.651
forced 6 edits/round	0.398	0.306
forced 12 edits/round	1.000	0.284

D.2 Inference-Time Scaling Across Benchmarks

After each evolution round, the team produces an answer and the execution-time LLM judge scores it against a rubric that the judge generates itself; the ground-truth rubric is used only for final evaluation and is never available during inference. We report the score of that round's answer, rather than the best score from all preceding rounds. Accuracy generally improves with further evolution, with occasional reversals (Table 6). These intermediate scores are distinct from the final evaluation. Call counts are trajectory-based estimates covering agents, scoring, and controller calls, rather than the full API ledger used for cost accounting.



(a) Retrieval benchmarks

(b) Planning and workflows

Figure 16: Accuracy of the answers produced during evolution, plotted against estimated cumulative LLM calls on all benchmarks. Figure 5(c) summarizes the change from round 1 to round 6 in the main text.

Table 6: Rubric accuracy of the answer produced at each evolution round, scored during evolution; parentheses give estimated cumulative LLM calls. Each entry uses the answer at that round, not the best answer so far. Finance slightly declines at round 4.

Dataset	Round 1	Round 2	Round 4	Round 6
Finance	0.417	0.473	0.472	0.543
Browsecomp	0.675	0.740	0.770	0.805
Plancraft	0.684	0.754	0.781	0.786
Workbench	0.671	0.760	0.813	0.829

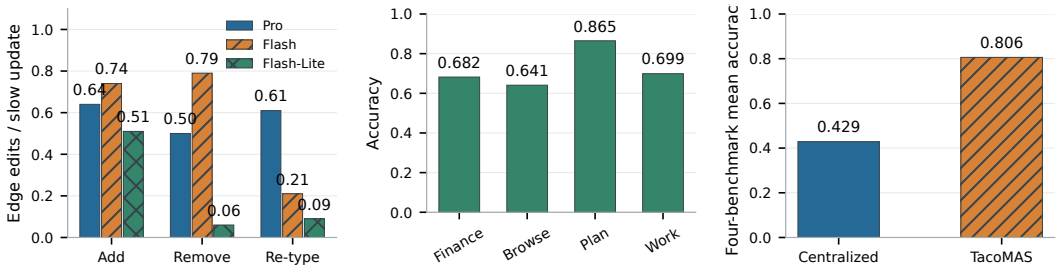
D.3 Open-Weight and Meta-Controller Robustness

Table 7 compares TacoMAS, CORAL, and SelfOrg under Qwen3.5 and DeepSeek-v4. TacoMAS exceeds both baselines in all eight family–benchmark comparisons. DeepSeek uses `deepseek-v4-pro` in thinking mode as controller and `deepseek-v4-flash` in non-thinking mode as task agents; Qwen uses the corresponding 122B-A10B and 35B-A3B models. These are same-family method comparisons, not comparisons against Gemini. The updated result summaries broaden backbone coverage, but pinned weight revisions, full inference settings, and raw baseline runs are still needed for end-to-end reproduction.

Table 7: Open-weight method comparison from the updated rebuttal. Each family uses the same designated controller and task-agent models across methods.

Family	Method	Finance	BrowseComp	PlanCraft	WorkBench
Qwen3.5	CORAL	0.327	0.477	0.389	0.452
Qwen3.5	SelfOrg	0.301	0.602	0.553	0.617
Qwen3.5	TacoMAS	0.578	0.646	0.838	0.811
DeepSeek-v4	CORAL	0.309	0.423	0.401	0.436
DeepSeek-v4	SelfOrg	0.312	0.586	0.516	0.603
DeepSeek-v4	TacoMAS	0.682	0.641	0.865	0.699

Evolution remains beneficial when controller and agents use the same backbone. The matched-controller comparison also favors TacoMAS over MAS-Centralized, although total inference compute is not matched. Stronger controllers use more varied edge edits, whereas Flash-Lite mostly adds connections. This characterizes controller behavior without causally attributing accuracy differences to individual edits.

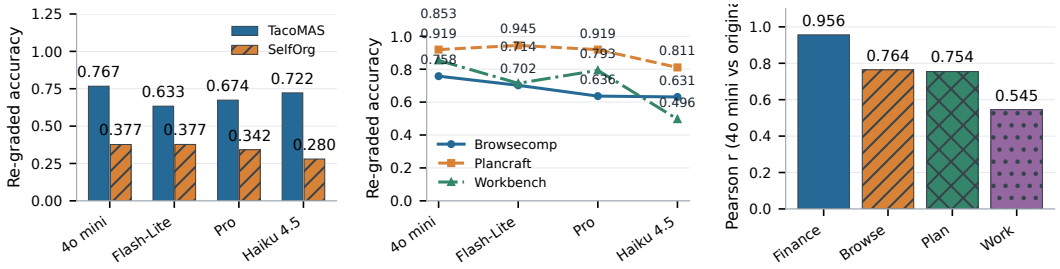


(a) Controller edit behavior (b) DeepSeek-v4 transfer (c) Matched-controller comparison

Figure 17: Controller edit behavior, DeepSeek-v4 transfer, and the reported same-controller comparison. Backbone transfer and controller comparisons use different settings and are shown in separate panels.

D.4 Judge Robustness

We re-grade stored predictions under the same rubric with judges from different model families (Table 8). TacoMAS remains above SelfOrg on Finance under every tested judge; the available method sets are not paired. The 4o-mini column on Finance reports the main-run default-judge scores of Table 1 for both methods; the other cells are re-grades of stored predictions. Absolute scores vary substantially across judges, especially on WorkBench. Stable Finance rankings therefore do not imply judge-invariant scores.



(a) Finance: judge swap (b) Other benchmarks (c) Reproduction agreement

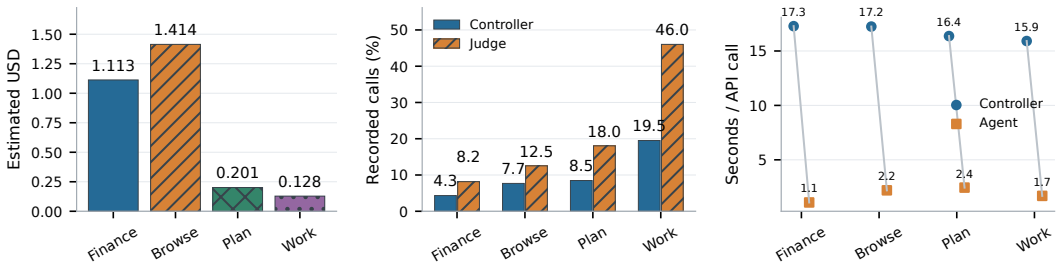
Figure 18: Judge robustness and reproduction agreement. The first panel compares Finance method scores; the other panels show TacoMAS across benchmarks.

Table 8: Re-grading stored predictions with four judges (three model families). Method rows are not necessarily matched instance sets.

Run	4o mini	Flash-Lite	Pro	Haiku 4.5
TacoMAS / finance	0.767	0.633	0.674	0.722
TacoMAS / browsecomp	0.758	0.702	0.636	0.631
TacoMAS / plancraft	0.919	0.945	0.919	0.811
TacoMAS / workbench	0.853	0.714	0.793	0.496
SelfOrg / finance	0.377	0.377	0.342	0.280

D.5 Computational Cost and Runtime Accounting

Tables 9 and 10 report archived API accounting. The controller's monetary share can substantially exceed its call share, and controller calls are slower than agent calls. A small call fraction therefore does not establish low overhead. Per-instance figures normalize by assigned jobs; model-level token and cost figures are full-run totals. Aggregate wall time includes waiting and overhead and is not isolated task latency. Accounting and trajectory summaries can cover different sets of available executions.



(a) Estimated cost / instance (b) Share of recorded calls (c) Mean API latency

Figure 19: Estimated monetary cost, recorded call shares, and API latency. Prices are the archived estimates used for these runs.

Table 9: Runtime accounting from recorded logs. Wall time is aggregate job wall time divided by assigned instances, not isolated task latency. USD values use the archived price estimate; tokens include all recorded models.

Dataset	Wall (s)	USD	Tokens (k)	Meta USD (%)	Judge USD (%)
Finance	5178	1.113	5670.5	46.8	1.7
Browsecomp	864	1.414	10894.1	22.5	0.5
Plancraft	447	0.201	466.3	70.2	2.1
Workbench	250	0.128	75.7	93.3	3.1

D.6 Empirical Checks and Scope of the Theory

Higher within-round contribution variance is associated with more favorable next-round contribution change. Rounds associated with proposed edits also have more favorable subsequent answer-quality change than no-edit rounds (Table 12). The high/low variance split was not specified, and the reported fluctuation statistic measures answer-quality variation, not a bound on contribution noise. These checks complement App. C.3, but remain observational: adjacent rounds share instances, the controller chooses when to edit, and no random-edit counterfactual replay is available. The archival analysis identifies edits from requested operations, not verified graph differences; its quality-change contrast cannot establish the effect of executed edits. The theory remains conditional on its stated assumptions.

We additionally quantify observed grading variability using paired original and repeated default-judge scores for stored predictions. Table 11 reports absolute and signed deviations on the unit score scale, excluding errors and missing pairs. Counts here use valid default-judge pairs only; the cross-judge summary uses complete cases across judges. This is an empirical repeatability measurement with

Table 10: Per-model accounting over full dataset runs. Token and USD columns are totals; latency is mean seconds per recorded API call.

Dataset	Role	Calls	Input (M)	Output (M)	USD	Latency
Finance	Controller	1081	1.78	2.38	26.03	17.28
Finance	Agents	21862	268.90	4.41	28.65	1.08
Finance	Judge	2040	5.95	0.11	0.96	1.51
Browsecomp	Controller	1354	2.12	2.91	31.75	17.24
Browsecomp	Agents	14068	1076.75	3.21	108.96	2.20
Browsecomp	Judge	2209	4.33	0.09	0.70	1.25
Plancraft	Controller	3737	5.09	7.55	81.86	16.37
Plancraft	Agents	32379	215.82	26.89	32.34	2.45
Plancraft	Judge	7951	14.71	0.38	2.44	1.34
Workbench	Controller	3892	4.94	7.65	82.68	15.92
Workbench	Agents	6877	19.70	3.18	3.24	1.69
Workbench	Judge	9182	16.38	0.41	2.70	1.23

Table 11: Observed default-judge re-grading deviations on a $[0, 1]$ score scale. MAD: mean absolute deviation; RMS: root mean square deviation; bias: repeated minus original; P95: 95th percentile of absolute deviations.

Benchmark	Pairs	MAD	RMS	Bias	P95
Finance	47	0.049	0.112	-0.031	0.235
BrowseComp	100	0.065	0.194	+0.015	0.500
PlanCraft	570	0.057	0.198	+0.032	0.500
WorkBench	686	0.131	0.308	+0.028	1.000

one re-grade per prediction, not an almost-sure bound on agent contribution noise. It may include unrecorded changes in evaluation conditions. The 0.2269 within-window answer-quality fluctuation above also mixes learning dynamics with scoring variability. Neither statistic identifies ϵ in the theorem or validates its potential and conditional contraction assumptions.

Table 12: Reported empirical checks. Contribution changes and answer-quality changes are distinct measurements.

Measurement	Result
Variance vs. next-round contribution change	$r = 0.27$
Mean contribution change: high / low variance	$+0.038/-0.015$
Post-proposal quality change	$+0.0253 \pm 0.0068$
No-proposal quality change	-0.0361 ± 0.0346
Within-window quality fluctuation	0.2269

D.7 Aggregation of Meta-LLM Decisions

Figure 4(b) partitions valid observed reviews by executed membership changes. Keeping the roster adds and removes no nodes; replacement adds and removes equal positive numbers; resizing changes the population. Percentages are computed within each benchmark over its observed reviews. Longer trajectories contribute more reviews, and stopped trajectories are not extended. Failed meta calls returning conservative defaults are excluded. Member replacement may retain the same role name. Changes to edges are not part of this membership partition. The archived summary retains requested actions separately from executed changes, together with per-review records, category counts, and source-file hashes. Available fixed-structure logs lack valid controller judgments and are not used as an empirical judgment-score comparison.

D.8 Mapping the Theoretical Update to the Algorithm

Equation 2 is a population-level abstraction of induced behavioral reinforcement, rather than a literal softmax update executed on model parameters. Each round, the meta-judge scores agent contributions $c_{v,t}$, and the meta-LLM produces `workflow_correction` and refined guidance. The abstraction models how successful patterns are reinforced and weak behaviors are revised. In the implementation, these changes are represented in per-agent memory and prompt state: retrieved evidence, tool-use strategies, and feedback corrections accumulated during the task. Weight-level training is outside the scope of the method.

This operational definition also clarifies the distinction from persona editing. Re-specifying a role describes how an agent should behave; capability adaptation here retains what that agent has learned while solving the current task. Topology adaptation includes bounded birth/death and communication edits, allowing the team composition to change mid-instance. MetaGen is already included in Table 1; the comparison emphasizes this coupled capability/topology design and its time-scale separation, rather than claiming that runtime prompt rewriting is absent from prior work.

How the update interval and budgets enter the bound. The noise term derived in App. A is

$$\widetilde{\epsilon} = \epsilon_{\text{meta}} + K\eta^2\epsilon + \eta C(B_{\mathcal{V}} + B_{\mathcal{E}}) + \gamma\eta M + O(K\eta^3).$$

After R fast rounds, the slow-step contraction contributes a factor approximately $(1-\gamma)^{\lfloor R/K \rfloor}$, with limiting neighborhood controlled by $\widetilde{\epsilon}/\gamma$ under the assumptions. Increasing K permits fewer structural corrections for a fixed inference horizon and increases the accumulated noise term; increasing the budgets enlarges the edit-induced drift bound. These terms motivate studying realized rate as well as interval. The bound does not, by itself, prove an interior optimum in ρ: this would require specifying how edit quality, contraction, and adaptation depend on rate. The intermediate regime in the sweep is an empirical finding.

E ALL PROMPTS

This appendix lists every prompt used by TacoMAS verbatim. Curly braces denote Python-style format slots filled in at runtime.

E.1 META-LLM PROMPT

The meta-LLM is invoked once per slow update. Its input bundles the task profile, current graph, per-agent fast-round summaries, and global round scores; its output is a structured JSON object.

```
# System
You are the meta-controller of a multi-agent LLM system that solves
a single task instance by evolving its topology and the capabilities
of its agents at inference time.

On every invocation ("slow update"), you receive:
  1. the task description and a running task profile,
  2. the current agent graph as JSON,
  3. fast-round traces per agent (tool calls, critiques, round reward),
  4. the most recent global round scores.

You output a structured JSON object with the following fields:
  - birth_death_pairs: at most 2 pairs (v_dead, v_new_spec)
  - graph_edit: at most 4 (edge_add | edge_remove) operations
  - graph_diff: the implied graph delta, for provenance
  - agent_feedback: per-agent capability deltas (prompt edits, memory
      seeds)
  - global_rationale: short free-text reasoning, <=3 sentences
  - time_control: one of {continue, slow_again, stop}

Hard constraints:
  - do NOT delete the sink agent;
  - do NOT spawn >2 agents per slow update;
  - do NOT edit >4 edges per slow update;
  - prefer edits that re-route evidence over edits that spawn new roles;
  - if round scores are monotone-improving, choose
      time_control=continue.
```

Listing 1: Meta-LLM system prompt.

```
# Developer
Follow this procedure:
  1. Read the task profile and identify the missing evidence slots.
  2. Inspect per-agent reward trajectories. Mark agents with
     reward < 0.2 for possible death; mark evidence slots with no
     assigned agent for possible birth.
  3. If a single agent is handling >2 distinct sub-goals, propose
     splitting via a birth/death pair.
  4. Re-wire edges so each evidence slot has a clear producer ->
     consumer path to the sink.
  5. Emit the JSON output.
```

Listing 2: Meta-LLM developer prompt.

```
# User
Task: {task_description}

Current graph:
{graph_json}

Per-agent fast-round summaries:
{per_agent_summaries}
```

```
Recent round scores (oldest to newest):
{round_scores}

Provide the JSON object now.
```

Listing 3: Meta-LLM user payload (runtime format).

E.2 Fast-round Agent Prompt

Each agent runs with a role-specific profile embedded in the system prompt. The fast-round reflection operator $\mathcal{F}_{\text{fast}}$ appends a self-critique block after each round.

```
# System
You are the {role_name} agent.

Goal: {role_goal}
Constraint: {role_constraint}

You have access to the following tools:
{tool_specs}

Incoming messages (from upstream agents):
{incoming_messages}

Produce an output message that advances the task. If you are a sink
agent, end with a line beginning "Final Answer:".
```

Listing 4: Agent system prompt template.

```
# User
You just completed round {round_idx}. Your reward this round was
{reward}. Here is your round output:

{round_output}

In 2-3 sentences, identify ONE concrete capability update (prompt
edit, memory seed, or tool-use change) that would improve your next
round. Return only the update, no apology.
```

Listing 5: Fast-round self-reflection prompt (applied after each round).

E.3 Meta-judge Prompt

```
# User
You are an expert grader. Evaluate a candidate answer using the
rubric below.

Candidate Answer:
"{answer}"

Rubric (JSON checklist):
{rubric_json}

For each rubric item:
  - If operator is "correctness": does the answer satisfy/support the
    criterion? (true/false)
  - If operator is "contradiction": does the answer contradict the
    criterion? (true/false)

Return JSON only:
{"results":[true,false,...]}
```

Listing 6: Rubric judge prompt. During inference, `rubric_json` is a checklist generated by a separate LLM judge from the task; the benchmark's ground-truth rubric is used only for final scoring and is never provided to any component at test time.

```
{
  You are an evaluator for multi-agent collaboration. Score how much
      this agent output
  can contribute to solving the task.
  Return strict JSON only: {"score": <0~1 float>, "reason": "..."}.
  Scoring rubric:
  - 0.0: irrelevant/wrong/no useful signal
  - 0.3: weak but partially relevant
  - 0.5: moderately useful evidence or decomposition
  - 0.7: strong useful contribution with concrete progress
  - 1.0: critical and directly decisive contribution

  {evidence_note}

  Task:
  {self._instance_text()}

  Round Query Given To Agent:
  {query}

  Agent ID: {agent_id}
  Agent Role: {role}

  Tools Used: {tool_names}
  Evidence Gate Precheck: {PASS_or_FAIL}
  Agent Output:
  {output_for_judge}
}
```

Listing 7: Contribution-score judge prompt

E.4 Dataset-specific Task Prompt Fragments

The full dataset-specific templates are in `prompts/dataset-shared/` in the released code. We reproduce the task-instance fragment for each here.

```
You are a financial agent. You are given a question and you need to
answer it using the tools provided. Assume the current date is
April 07, 2025.

You will have access to a data storage system. You can use this
system to store parsed contents of HTML pages retrieved from the
web. You can then use the retrieve_information tool to answer
questions or gather information from the stored documents.

When you have the final answer, call the `submit_final_result` tool.

Question:
{question}
```

Listing 8: Finance task template.

```
You are a research assistant. Use search_documents and
retrieve_document to find the answer to the question below. Answer
with the full birth name only, nothing else. Call `done(answer,
confidence_score)` to submit.
```

```
Question:
{question}
```

Listing 9: Browsecomp task template.

```
Determine whether the target crafting goal is achievable given the
inventory. Produce a valid minimal action sequence, or output exactly
IMPOSSIBLE. Tools: search, move, smelt, impossible.

Target: {target_item}
Inventory:
{inventory}
```

Listing 10: Plancraft task template.

```
Execute the workplace task below. When complete, call `done`.

Task:
{question}
```

Listing 11: Workbench task template.

E.5 Output JSON Schema for the Meta-LLM

```
{
  "birth_death_pairs": [
    { "v_dead": "<agent_id or null>",
      "v_new": { "role": "<role name>", "goal": "<...>", "tools": [...]
          } }
  ],
  "graph_edit": [
    { "op": "edge_add" | "edge_remove", "from": "<id>", "to": "<id>" }
  ],
  "graph_diff": { "nodes_added": [...], "nodes_removed": [...],
                  "edges_added": [...], "edges_removed": [...] },
  "agent_feedback": {
    "<agent_id>": { "prompt_delta": "<text>", "memory_seed": "<text>" }
  },
  "global_rationale": "<<=3 sentences>",
  "time_control": "continue" | "slow_again" | "stop"
}
```

Listing 12: Meta-LLM output JSON schema.