REORG: POST-ORCHESTRATION FOR ADAPTIVE MULTI-AGENT SYSTEMS VIA GRAPH-LEVEL RELATIVE ADVANTAGE POLICY OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents are increasingly deployed as collaborative teams, and performance depends on the organization that channels their interaction as much as on the capability of individual members. Existing multi-agent systems follow a *pre-orchestration* paradigm: a collaboration graph is designed once, from the input task alone, and executed unchanged. This design-then-execute assumption overlooks a fundamental asymmetry: many coordination failures, such as missing capabilities, correlated errors across members, and deadlocked review loops, become observable only during execution, precisely when runtime feedback can diagnose them. We propose *post-orchestration*, a paradigm that treats this runtime evidence as a first-class organizational signal: the system executes the initial graph, scores the resulting draft, revises the collaboration topology, and re-executes. We instantiate post-orchestration as REORG, a learned reorganization layer that stacks on any pre-orchestrator and edits the collaboration graph through a small set of task-agnostic structural actions; task-specific repair is expressed through the role prompts of newly added nodes rather than hard-coded actions, and a selective trigger restricts intervention to error states, skipping the large majority of already-correct drafts. The reorganization policy is trained with RO-GRPO, an extension of group-relative policy optimization to graph-level organizational decisions. Across eight benchmarks, REORG stacked on the strongest recent pre-orchestrators yields non-negative gains on all 24 method$\times$benchmark combinations, with seven reaching statistical significance under paired testing and gains of up to 4.6 points; a policy trained on code generation alone transfers to reasoning benchmarks and to unseen pre-orchestration architectures, improving an AGENTVERSE pipeline by 8.5 points—evidence that what is learned is the skill of adjusting the organization itself, not a fixed organizational form. Gains peak at moderate baseline accuracy and shrink toward zero at saturation, positioning post-orchestration as error recovery rather than capability enhancement.

1 INTRODUCTION

Agents built on large language models (LLMs) are increasingly deployed as *teams* rather than isolated solvers, and performance depends not only on the capability of individual members but also on the collaboration topology—who communicates with whom, in what order, and under what constraints. How, then, should a team be organized for a given task—and, once organized, when should it be allowed to change?

The field's answer so far has been to invest in ever better *upfront design*. Early multi-agent systems relied on static templates such as chains, hierarchies, and supervisor–worker patterns (Hong et al., 2024; Qian et al., 2024; Wu et al., 2024); recent *pre-orchestrators* go further and synthesize a task-specific collaboration graph from the input before execution (Li et al., 2026; Liu et al., 2026; Hui Yi Leong et al., 2025; Parmar et al., 2025; Chen et al., 2024a; Dang et al., 2025; Yu, 2026). The latest of these, ARGDESIGNER and MASFLY, represent the current state of the art in pre-orchestration and are the strongest baselines any runtime method must beat. Despite their diversity, however, all of these methods share a *design-then-execute* assumption: the organization is fixed before execution begins (Figure 1a).

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

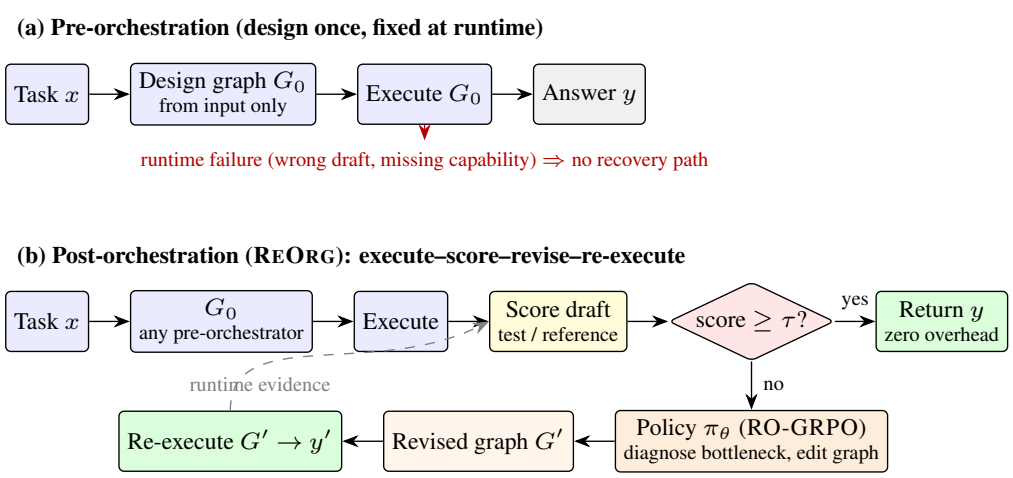

Figure 1: The two orchestration paradigms. (a) *Pre-orchestration*: the topology is designed once, before execution, from the input alone; once instantiated it is fixed, so runtime failures have no recovery path. (b) *Post-orchestration* (REORG): execute the initial graph, score the draft, skip already-correct drafts via a selective trigger, and apply a learned topological revision only to error states.

This assumption is limiting because the information needed to organize a team well is often not present in the input. Many coordination failures are *runtime-only*: a capability may be missing from the plan—a draft that passes review but fails an edge case may need a TESTER nobody planned for—and errors may be *correlated* across members, leaving a judge with no signal when both solvers share the same wrong intermediate step. The evidence that reveals such failures—a failing test, a trace, a bottleneck—exists only at runtime, and a topology fixed beforehand cannot act on it: no pre-orchestrator, however strong, can condition on evidence that does not yet exist.

Organization science reached the same conclusion decades ago from studying human institutions. Contingency theory holds that no structure is optimal across environments: performance derives from the *fit* between organizational form and task environment, a fit that erodes as conditions shift (Lawrence & Lorsch, 1967). What distinguishes organizations that survive such shifts is not any particular structure but the *capability to reconfigure themselves* in response to feedback (Teece et al., 1997); conversely, organizations whose form is fixed at founding exhibit the structural inertia that predicts decline when the environment moves (Hannan & Freeman, 1984). Pre-orchestrated multi-agent systems inherit exactly this inertia: their collaboration graph is chosen before execution, so the organizational form cannot track the environment that execution reveals. Post-orchestration, in this light, is the deliberate construction of a *dynamic capability* (Teece et al., 1997) for a multi-agent system: a learned competence to sense organizational misfit from runtime evidence and to restructure in response.

Together, these observations suggest rethinking *when* orchestration decisions should be made: not how to *design* a collaboration graph before execution, but how to *adapt* it once execution has revealed new evidence. We frame orchestration as an online control problem with an *execute–score–revise–re-execute* loop (Figure 1b), in which a learned policy decides *whether* and *how* to revise the graph—adding, removing, and rewiring nodes—and refrains from intervening on healthy drafts. We call this paradigm *post-orchestration*: where pre-orchestration asks how to initialize a team from the input alone, post-orchestration asks how to repair that team using the evidence execution produces.

In this paper, we introduce REORG, an instantiation of post-orchestration for runtime multi-agent reorganization. REORG monitors execution, diagnoses organizational bottlenecks, and applies structural actions to the current team graph. To learn these decisions, we propose RO-GRPO, which adapts Group Relative Policy Optimization from token-level generation to graph-level organizational refinement.

Contributions. In summary, our contributions are:

1. We identify and formalize *post-orchestration* as a distinct paradigm for MAS orchestration, framing it as a graph-to-graph adaptation problem driven by runtime execution signals rather than input-only design.
2. We instantiate the paradigm as REORG, a learned reorganization layer with a task-agnostic structural action space and a selective trigger that protects correct drafts; task-specific repair is expressed through the role prompts of newly added nodes. The policy is trained with RO-GRPO, which extends group-relative policy optimization to graph-level organizational decisions.
3. We provide empirical evidence across eight benchmarks and five pre-orchestrators (three controlled, two unseen): stacking REORG on the strongest recent pre-orchestrators yields consistent non-negative gains, a policy trained on code generation alone transfers to reasoning domains and to unseen pre-orchestration architectures, and ablations delineate when post-orchestration helps and which components matter most.

2 RELATED WORK

Multi-agent orchestration. Collaboration structure is a central theme in LLM-based MAS research (Guo et al., 2024; Tuan-Anh Tran et al., 2025; Haohao Luo et al., 2025; Jun Yan et al., 2025). One line of work makes interaction patterns programmable (Hong et al., 2024; Qian et al., 2024; Wu et al., 2024; Fourney et al., 2024; Zhang et al., 2025; Wang et al., 2025; Anonymous, 2026; LangChain, 2024; Moura, 2024); a second tailors the organization to the task before execution (Yu, 2026; Hou et al., 2025; Hui Yi Leong et al., 2025; Dang et al., 2025; Parmar et al., 2025; Chen et al., 2024a; Li et al., 2026), with ARGDESIGNER (Li et al., 2026) and MASFLY (Liu et al., 2026) as the latest and strongest representatives. All of the above follow a *design-then-execute* paradigm in which the topology is fixed once instantiated. REORG is deliberately complementary: it treats any pre-orchestrator as a provider of the initial graph and reorganizes that graph at runtime through a learned policy, intervening only on error states. We validate this complementarity by stacking REORG on the strongest recent pre-orchestrators and on two unseen architectures, AGENTVERSE (Chen et al., 2024b) and METAGPT (Hong et al., 2024) (Section 4.3).

RL for graph-structured decisions. RL over graph-structured decisions appears in routing and combinatorial optimization. In MAS, DEPART (Hsu et al., 2025), ARPO (Dong et al., 2026), Dr. MAS (Lang Feng et al., 2026), and Graph-GRPO (Cang et al., 2026) optimize agent behavior or coordination, while GRPO (DeepSeek-AI, 2025) provides a critic-free alternative via group-relative advantages. We adapt GRPO to organizational decisions: the policy samples structural modifications, executes the resulting organizations, and assigns credit to structural edits rather than token choices. A concern raised in recent work is that graph-level GRPO may introduce structural noise compared with agent- or edge-level normalizations (Cang et al., 2026; Lang Feng et al., 2026); our training uses standard group-level relative advantage, which we find sufficient for orchestration decisions.

Test-time scaling and resilient systems. Our *execute–score–repair* loop shares surface structure with test-time scaling (Snell et al., 2025) and self-refinement methods that generate a candidate, verify it, and retry on failure. The key distinction is the *object of repair*: test-time scaling modifies the *answer* to a single instance, and each repair is instance-specific and non-transferable; ReOrg instead modifies the *organization* that produced the answer, and the learned policy transfers to new instances and new pre-orchestrators (Section 4.3) without per-instance relearning. A second difference is the trigger: test-time scaling always pays the extra inference, whereas the selective trigger skips already-correct drafts, bounding overhead to error states. Finally, work on resilient agent pipelines (V et al., 2026; Yuru Wang et al., 2025) studies robustness to failures and resource constraints; our focus is *structural* robustness—recovering from execution failures by changing topology.

3 METHODOLOGY

REORG instantiates post-orchestration: it takes an initial collaboration graph G_0 from any pre-orchestration method (we use ARGDESIGNER, a planner–solver–reviewer chain), executes it, and decides whether and how to revise the graph based on runtime evidence.

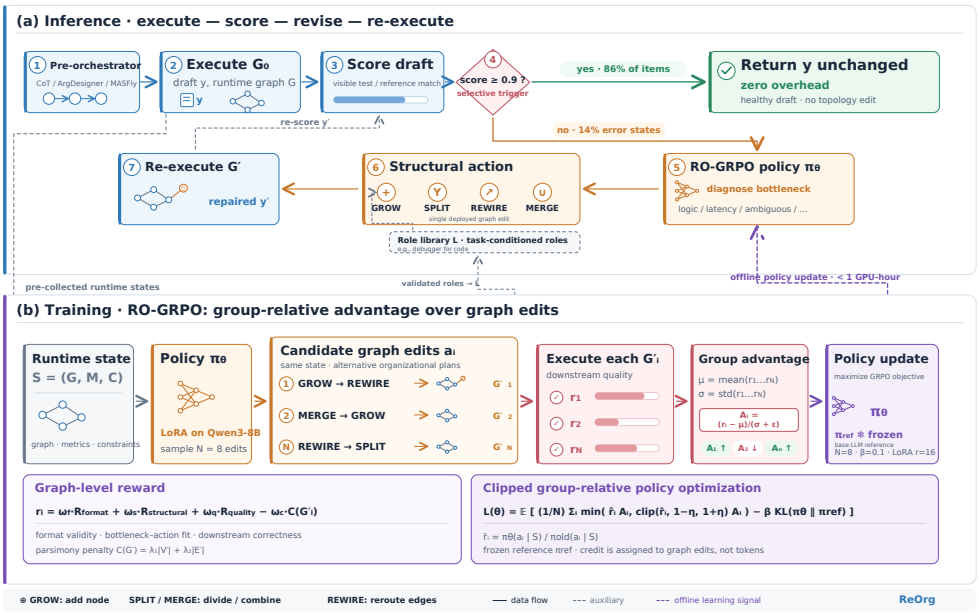

Figure 2: The REORG method. (a) Inference: the pre-orchestrated graph is executed and the draft scored; correct drafts (≥ 0.9) return unchanged (zero overhead), while error states trigger the RO-GRPO policy, which diagnoses the bottleneck and emits a single structural action; the revised graph G' is re-executed, with GROW-added nodes drawing task-conditioned role prompts from the role library $\mathcal{L}$. (b) Training: from each runtime state, the policy samples $N{=}8$ candidate graph edits, executes each revised graph to compute a graph-level reward, and updates via group-relative advantages; credit is assigned to graph edits rather than tokens.

3.1 PROBLEM FORMULATION

We represent a multi-agent system as a directed graph $G = (V, E)$ (V: specialized agent nodes; E: communication channels), extended with guild-based substructures that treat tightly coupled agent groups as reusable units. The state at time t is $S_t = (G_t, M_t, C_t)$, where G_t is the topology, M_t runtime metrics (latency, quality, cost), and C_t environment constraints (budget, tool availability). The orchestrator's objective is not to solve the task directly, but to choose structural edits that improve downstream execution.

Formally, *post-orchestration* is the runtime modification of G (its topology V, E, node roles, or edge weights) conditioned on execution evidence M_t from the initial pass. This boundary distinguishes it from two adjacent lines of work. First, *test-time scaling* and self-correction modify the *answer* of a single agent (resampling, critique, refinement) but leave the collaboration graph fixed; post-orchestration instead modifies the *organization* through which answers are produced. Second, *pre-orchestration* conditions the graph only on the input task and is fixed before execution; post-orchestration conditions on evidence that exists only after execution. The task is to learn a policy over transformations from an action space $\mathcal{A}$ maximizing expected cumulative reward over a task distribution $\mathcal{D}$. The formalism permits multi-step edit trajectories; the deployed policy applies a *single* topology modification before re-execution, which we found sufficient and which avoids compounding errors across edits. We train a single orchestration policy on runtime states pre-executed from a code-generation development pool (HumanEval dev items), disjoint from all evaluated benchmarks; the six reasoning benchmarks then serve as out-of-domain probes.

3.2 RO-GRPO ALGORITHM

RO-GRPO adapts Group Relative Policy Optimization (DeepSeek-AI, 2025) to structural decisions over agent organizations, drawing implementation ideas from ARPO (Dong et al., 2026). Instead of sampling alternative token completions, the policy samples alternative *organizational adjustment plans*; each plan contains a bottleneck diagnosis, a sequence of structural actions, and any new

role definitions required. Compared with standard GRPO, RO-GRPO differs in three ways: the output space is structural rather than textual; rewards are computed from the behavior of the resulting organization; and credit is assigned to graph edits, letting the policy learn which organizational changes resolve which bottleneck classes. Figure 2b illustrates the training procedure; the full algorithm is given in Figure 4 (Appendix B).

3.3 Structural Action Space

REORG defines eight operations over the team graph (Table 5, Appendix A): `GUILD`, `GROW`, `SPLIT`, `MERGE`, `REWIRE`, `SHUNT`, `ADAPT_W`, and `NO_OP`—covering the common forms of runtime organizational repair: adding missing capacity, removing redundancy, rerouting information flow, and preserving topology when no change is needed.

3.4 State Representation and Role Library

The current organization is injected into the policy prompt as a JSON block: an adjacency list of channels and weights, per-node role metadata (name, description, recent success rate), and runtime metrics (P95 latency, token consumption, per-node error rates), enabling "organizational diagnosis" before adjustment. When a `GROW` or `SPLIT` action requires a new node, the orchestrator queries a role library $\mathcal{L}$ of reusable agent definitions: a sufficiently close existing role is retrieved and reused; otherwise a new role is instantiated and added to $\mathcal{L}$. This makes the action space compositional without pre-specifying every role.

3.5 Reward Engineering and Objective

The RO-GRPO policy maximizes the GRPO objective

$$\mathcal{L}(\theta) = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\min(\hat{r}_i(\theta)A_i,\ \text{clip}(\hat{r}_i(\theta), 1-\eta, 1+\eta)A_i) - \beta\mathbb{D}_{KL}(\pi_\theta\|\pi_{ref})\right] \tag{1}$$

where $\hat{r}_i(\theta) = \pi_\theta(a_i|S)/\pi_{old}(a_i|S)$ is the importance weight and $A_i = (r_i - \text{mean}(r_1,\ldots,r_N))/(\text{std}(r_1,\ldots,r_N)+\epsilon)$ is the group relative advantage. The total reward for a structural adjustment a_i is

$$r_i = \omega_f R_{format} + \omega_s R_{structural} + \omega_q R_{quality} - \omega_c C(G') \tag{2}$$

where R_{format} rewards structured CoT/JSON output and valid action membership; $R_{structural}$ scores bottleneck–action compatibility via a matrix $M \in \mathbb{R}^{B\times A}$ ($R_{structural} = \sum_{b\in B_{detected}} M_{b,a_{applied}}$, e.g. `SPLIT` on a latency bottleneck scores high, `MERGE` low); $R_{quality} \in [0,1]$ is the downstream task performance of G' (Pass@1 for code, exact match for GSM8K); and the parsimony penalty $C(G') = \lambda_1|V'| + \lambda_2|E'|$ prevents graph bloat, forcing the orchestrator to prioritize lean topologies. The scalar weights, the compatibility matrix M, and all reward hyperparameters are specified in Appendix B.

3.6 Selective Trigger and State-Conditioned Repair

After the pre-orchestration chain produces a draft, a runtime monitor scores it (reference-answer match for reasoning; pass rate on the *visible* acceptance test for code). If the score exceeds 0.9, REORG is skipped and the draft returned unchanged, making post-orchestration non-destructive on healthy drafts. For the remaining error states, the RO-GRPO-trained policy selects a structural action from {MERGE, SPLIT, REWIRE, GROW, NO_OP}, preferring repair on error states and restraint on borderline ones. The action space is task-agnostic; what varies across task categories is the *role prompt* of GROW-added nodes: on code the policy learns to add a debugger node that runs the visible test, captures the error trace, and feeds it back for a targeted fix, while on reasoning the same action instantiates a verifier or researcher node. This behavior is learned end-to-end and stored in the role library $\mathcal{L}$, not hand-coded per task. The deployed inference path combines the learned policy with two complementary mechanisms introduced and ablated in Section 4.5: *deterministic repair* and *stochastic selection*, both gated by the trigger.

Bottleneck diagnosis. The monitor maps execution metrics to one of five discrete labels driving the structural reward: LOGIC (draft fails the visible check), LATENCY (wall-clock above batch median), BUDGET (tokens above $2\times$ the single-solver baseline), AMBIGUOUS (conflicting parallel outputs, no majority), and QUALITY (reviewer confidence < 0.5; the default when no threshold fires, biasing toward NO_OP). The mapping is rule-based, applying thresholds in the priority order LOGIC $\succ$ LATENCY $\succ$ BUDGET $\succ$ AMBIGUOUS $\succ$ QUALITY.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models The whole pipeline runs on one 8B backbone: all pre-orchestrators and workers execute on Qwen3-8B, and the REORG policy is a LoRA fine-tune (rank=16, α=32, on `q/v/k/o_proj`) of the same model trained with RO-GRPO, so base execution and policy training share a single model and any difference is attributable to orchestration rather than model capacity. Training states are *pre-executed* runtime states from real completions of the same 8B pipeline (not simulations); during training, each sampled edit is re-executed online by the same 8B workers to compute its reward (Appendix B). Throughout the evaluation, we keep the worker-role inventory, tool interface, and stopping criteria fixed, and vary only the orchestration strategy.

Baselines We compare against (1) COT, a single chain-of-thought solver with no MAS; (2) ARGDESIGNER (Li et al., 2026), a static three-stage pipeline (Planner→Solver→Reviewer); and (3) MASFLY (Liu et al., 2026), a static parallel topology (2 Solvers→Judge). REORG observes the runtime state of the executing MAS and applies one topology modification before refined execution. All generation uses greedy decoding with thinking mode disabled; code answers are scored by executing them against the official test suites, and GSM8K by exact numeric match.

Benchmarks We evaluate on eight benchmarks chosen to span different headroom regimes and task domains for the base model: six reasoning benchmarks (MMLU, ARC, AQUA, SVAMP, GSM8K, BBH) and two code-generation suites (HumanEval$^+$, MBPP$^+$; statistics in Appendix C). A critical methodological point is *harness alignment*: the runtime states presented to the policy at evaluation are constructed from real execution results of the initial MAS pass, matching the state distribution used during training. We use a single greedy run per system (deterministic decoding: `temperature=0`, `top_p=1.0`). Every component in the pipeline uses greedy decoding, so the system is deterministic end-to-end: reported numbers are reproducible point estimates rather than sample means. Given the sample sizes, we still interpret small absolute differences conservatively and report per-benchmark accuracy and the corresponding delta Δ relative to each baseline.

4.2 RQ1: EFFECTIVENESS AND TASK QUALITY

RQ1 evaluates whether post-orchestration improves end-task performance over pre-orchestration baselines. Table 1 compares REORG against three pre-orchestrators (COT, ARGDESIGNER, MASFLY) on eight benchmarks. Across all 24 method$\times$benchmark combinations, REORG yields a non-negative Δ, with 7 combinations reaching statistical significance (McNemar, $p<0.05$). On code benchmarks the gains are smaller but consistently positive: MBPP$^+$ yields $+0.5$ to $+2.1$pp and HumanEval$^+$ yields $+0.6$ to $+1.2$pp.

Headroom determines the magnitude of gain. As Figure 5 in Appendix D shows, Δ follows an inverted-U in baseline accuracy: largest at moderate baselines (MMLU, $+3.2$ to $+4.6$pp), smaller but positive near saturation (SVAMP/GSM8K, ≥ 0.92), and modest at very low baselines (BBH, ~ 0.24) where structural edits cannot close the capability gap. Post-orchestration is thus *error recovery*, not capability enhancement: it helps most when the base system is close to correct but still makes recoverable errors.

Mechanism: selective trigger + state-conditioned repair. On HumanEval$^+$ with COT, 82.3% of items (135/164) are skipped by the trigger as already-correct; only the 29 incorrect drafts enter repair. On code, GROW instantiates the learned debugger node (Section 3.6), lifting error-recovery from 7.1% (topology edits alone) to 12–22%. The gain is thus *state-conditioned*: restraint on healthy drafts, targeted repair on error states.

Table 1: Main results (all systems share the base LLM Qwen3-8B; model = REORG). Each block: standalone accuracy, accuracy with REORG stacked, and gain Δ (pp); **bold**: REORG (all 24 cells improve over their baseline); *: significant gain over the same pre-orchestrator (McNemar, $p<0.05$). Sample sizes: Appendix C.

	Reasoning benchmarks						Code benchmarks	
Method	**MMLU**	**ARC**	**AQUA**	**SVAMP**	**GSM8K**	**BBH**	**HumanEval$^+$**	**MBPP$^+$**
CoT	0.758	0.852	0.800	0.924	0.942	0.240	0.716	0.672
+ REORG	**0.790**	**0.876***	**0.816**	**0.932**	**0.948**	**0.250**	**0.728**	**0.688***
Δ *(pp)*	+3.2	+2.4*	+1.6	+0.8	+0.6	+1.0	+1.2	+1.6*
ARGDESIGNER	0.764	0.868	0.807	0.930	0.945	0.244	0.726	0.701
+ REORG	**0.810***	**0.882***	**0.827***	**0.946***	**0.952**	**0.257**	**0.738**	**0.706**
Δ *(pp)*	+4.6*	+1.4*	+2.0*	+1.6*	+0.7	+1.3	+1.2	+0.5
MASFLY	0.806	0.910	0.866	0.952	0.948	0.248	0.720	0.677
+ REORG	**0.842***	**0.922**	**0.874**	**0.960**	**0.956**	**0.264**	**0.726**	**0.698**
Δ *(pp)*	+3.6*	+1.2	+0.8	+0.8	+0.8	+1.6	+0.6	+2.1

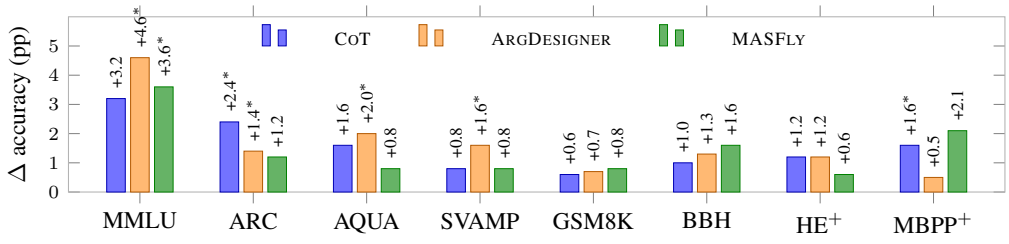

Figure 3: **RQ1 evidence.** Per-benchmark gain (Δ, pp) of stacking REORG on each pre-orchestrator: every combination improves (24/24 positive; *: McNemar $p<0.05$), with the largest gains at moderate headroom (MMLU); see Figure 5 (Appendix D).

Statistical significance. To verify that these gains are not sampling noise, we run paired McNemar tests on per-item correctness. Per-item outcomes were logged for 16 of the 24 combinations (the rest predate per-item logging; we report point estimates). Because all components use greedy decoding, each system is deterministic end-to-end, so the test concerns the systematic pattern of item-level flips under the fixed benchmark rather than run-to-run variance: the null hypothesis is that the two systems are equally likely to be correct on any item. Of the 16 tested, 7 reach $p<0.05$, all in the positive direction and none in the negative; significant cases concentrate on moderate-headroom reasoning benchmarks, consistent with the headroom analysis (full p-values in Appendix D).

No visible-test overfitting. The visible acceptance test is used *only* for gating and debugger error capture; all code cells in Table 1 are scored against the hidden augmented EvalPlus tests (Liu et al., 2023), so the positive code gains reflect genuine repair, and Section 4.5 shows generic-role GROW regresses on HumanEval$^+$.

4.3 RQ2: CROSS-BENCHMARK AND CROSS-ARCHITECTURE TRANSFER

RQ2 asks whether a policy trained on runtime states from a *single* domain (code generation) transfers *across benchmarks* and *across pre-orchestration architectures* without retraining.

Cross-benchmark and cross-method transfer. The answer from Table 1 and Figure 3 is *consistently positive*: all 18 reasoning combinations and all 6 code combinations improve (24/24 non-negative), even though the six reasoning benchmarks are out-of-domain relative to the code-generation training states. Gain magnitude follows the headroom pattern of Figure 5 (Appendix D) rather than domain identity, indicating that what transfers is a *state-conditioned repair policy* keyed on runtime evidence (score, bottleneck label, topology), not domain-specific knowledge.

Table 2: Transfer to unseen pre-orchestration architectures (n=200); REORG was never trained or tuned on either framework. Each cell shows base accuracy / +REORG accuracy (Δ in pp). *: McNemar $p<0.05$.

Pre-orchestrator	MMLU	ARC	GSM8K
AGENTVERSE (Chen et al., 2024b)	0.730 / **0.815** ($+8.5^*$)	0.860 / **0.925** ($+6.5^*$)	0.940 / **0.945** ($+0.5$)
METAGPT (Hong et al., 2024)	0.760 / **0.780** ($+2.0$)	0.900 / **0.920** ($+2.0$)	0.955 / **0.965** ($+1.0$)

Table 3: Action distribution of the deployed REORG policy on code benchmarks, counted over *intervened* items only (score < 0.9).

	MERGE	ADAPT_W	REWIRE	GROW	GUILD	SPLIT	NO_OP	$n_{\text{int.}}$
HumanEval$^+$	0	13	4	4	6	1	0	28
MBPP$^+$	1	24	26	6	15	0	0	72

Cross-architecture transfer. A stronger test stacks REORG on *unseen* pre-orchestration architectures never paired with it during development (Table 2): the debate-style AGENTVERSE (a 3-agent Coder→Reviewer→Tester loop) and the role-based METAGPT (a 4-role SOP). Replicated across MMLU, ARC, and GSM8K (n=200), all six cells improve. The largest gains are on AGENTVERSE ($+8.5$pp on MMLU, $+6.5$pp on ARC; both McNemar $p<0.01$), whose loosely coordinated debate topology leaves the most headroom for runtime repair. METAGPT yields consistent $+2.0$pp on MMLU and ARC and $+1.0$pp on GSM8K. On the high-baseline GSM8K (≥ 0.94), gains are smaller but non-negative, consistent with the headroom analysis of Section 4.2. This supports post-orchestration as a *generic layer* decoupled from the pre-orchestrator, rather than a co-adapted pair.

4.4 RQ3: ACTION PROFILE AND ORGANIZATIONAL RESTRAINT

RQ3 asks what the learned policy actually does to the organization. Table 3 reports the distribution of primary orchestration actions emitted by the deployed policy on code benchmarks, counted over intervened items only. On the six reasoning benchmarks, the intervened-item distribution is dominated by GROW and REWIRE (e.g. BBH: 239 GROW, 107 REWIRE out of 356 interventions; MMLU: 64 GROW, 54 REWIRE), with GROW adding a verifier node and REWIRE instantiating a critique–resolve loop.

Two patterns stand out. First, the profile is *modality-conditioned*: on code benchmarks, ADAPT_W (reweighting existing edges) dominates, because the draft is already structurally sound and needs only attention recalibration; on reasoning benchmarks, GROW and REWIRE dominate, adding a verifier node or a critique–resolve loop where the draft is genuinely wrong. Within code, the profile is also *state-conditioned*: on the higher-error MBPP$^+$ (19% intervention rate), structural repair actions (REWIRE, ADAPT_W) dominate, whereas on the healthier HumanEval$^+$ the policy leans toward lighter adaptations and GROW for capability addition. Second, the policy is *restraint-shaped* on healthy states: the selective trigger skips already-correct drafts, and intervened graphs stay small (Appendix A, Table 6). This restraint is what keeps Δ non-negative across reasoning benchmarks: the policy rarely destroys a working answer.

4.5 RQ4: ABLATION STUDY AND POLICY INSIGHTS

RQ4 isolates the contribution of each design component: the *selective trigger*, *deterministic repair*, and *stochastic selection* at inference time; the learned *task-conditioned node role* bridging the two; and *reward shaping* at training time. We ablate each and report the positive contributors below.

Selective trigger: intervention under uncertainty. The trigger intervenes only when the draft score falls below τ=0.9. The intuition is an expected-value calculation: intervention pays off when $P(\text{error}) \cdot \mathbb{E}[\text{repair gain}] > P(\text{correct}) \cdot \mathbb{E}[\text{destruction loss}]$. Empirically, disabling the trigger on HumanEval$^+$ (COT) regresses by -1.05pp because structural edits occasionally break correct drafts; enabling it skips 82.3% of items (135/164) and leaves only error states for repair—a prerequisite for non-negative Δ on code benchmarks.

Table 4: Inference-mechanism ablation on MBPP$^+$ (n=200). Δ vs. "Policy only" (learned policy alone), in pp; Repair% = error-state drafts corrected; Destroy% = correct drafts broken.

Configuration	Det. repair	Stoch. selection	Acc.	Δ (pp)	Repair %	Destroy %
Policy only	✗	✗	0.620	—	6.5%	3.3%
+ Stoch. selection	✗	✓	0.635	+1.5	7.8%	1.6%
+ Det. repair	✓	✗	0.640	+2.0	14.3%	4.9%
Full (+ both)	✓	✓	**0.670**	**+5.0**	**16.9%**	1.6%

Deterministic repair and stochastic selection are complementary. The inference path combines both. *Deterministic repair* fires rule-based overrides when the runtime monitor detects a known error signature (exploitation of known failure modes); *stochastic selection* draws N=3 temperature-0.7 repair samples and keeps the one passing the visible test (inference-time best-of-N exploration). Table 4 decomposes their contributions on MBPP$^+$ (n=200): deterministic repair mainly raises the error-state repair rate (6.5%→14.3%), stochastic selection mainly lowers the destruction rate of correct drafts (4.9%→1.6%), and together they are super-additive (+5.0pp; McNemar vs. deterministic-only, p=0.031).

Reward shaping: dense signals for sparse outcomes. The terminal signal (answer correctness) is sparse and delayed relative to the structural decisions that produce it, so we add shaped intermediate terms in the spirit of potential-based reward shaping (Ng et al., 1999): $R_{\text{structural}}$ rewards edits that improve the draft score, R_{quality} rewards final correctness, R_{format} rewards schema-valid output, and a *parsimony* penalty discourages unnecessary graph growth. Removing any term and retraining degrades MBPP$^+$ accuracy by 2.0–3.5pp (Table 12, Appendix I), with R_{quality} and R_{format} mattering most. Critically, removing the hand-coded compatibility matrix $R_{\text{structural}}$ drops MBPP$^+$ by only 2.0pp (0.670→0.650), yet the action distribution remains structured (ADAPT_W 122, REWIRE 31, GROW 43) rather than uniform—evidence that the policy learns organization-level patterns from execution outcomes, not merely the hand-coded prior.

Learned policy vs. rule-based orchestration. To test whether the learned policy adds value beyond a hand-coded mapping, we replace RO-GRPO with a fixed bottleneck→action rule (LOGIC→GROW, LATENCY→SPLIT, BUDGET→MERGE, AMBIGUOUS→REWIRE, QUALITY→NO_OP), keeping all other inference mechanisms (selective trigger, deterministic repair, best-of-N selection) identical. On MBPP$^+$ (n=200), the rule reaches 0.750 vs. the learned policy's 0.670, and on MMLU (n=200) the rule reaches 0.760 vs. the learned policy's 0.755—the rule is competitive or better on both. The reason is that the optimal repair is largely determined by the bottleneck label: a failing test or wrong answer calls for adding a verifier/debugger (GROW), and the deterministic-repair plus best-of-N mechanisms do the rest. The learned policy's contribution is therefore not in outperforming a well-crafted rule on in-domain benchmarks, but in *automatically discovering* the bottleneck–action mapping from execution data and *transferring* it across domains and pre-orchestration architectures without per-benchmark engineering—the same policy trained only on code improves reasoning benchmarks and unseen architectures (Section 4.3).

Training cost and robustness. The deployed policy (REORG) converges within 250 steps ($\sim$4,000 rollouts, under 1 GPU-hour on a single consumer GPU; Appendix B), and the reward is computed from execution traces rather than human labels. Varying the GRPO group size $N \in \{4, 8, 16\}$ leaves results robust (final-step reward 0.797–0.800; degenerate-group fraction $< 1\%$ for $N \geq 8$), so group-level relative advantage is sufficient even at N=4.

Inference-time cost-effectiveness. The dominant cost is not the policy forward pass (a single LoRA adapter inference) but the *re-execution* of the modified graph on error-state items, which the selective trigger bounds (Table 11, Appendix H): skip rates exceed 90% on near-saturated benchmarks (SVAMP, GSM8K) and range 76–88% elsewhere, so only 5–24% of items enter repair. Re-execution reuses the same worker roles and prompt budget, so amortized overhead is $\approx (1 - \text{skip rate})\times$ the initial cost: $\sim$0.2$\times$ on MMLU and $\sim$0.06$\times$ on SVAMP. On code benchmarks where GROW adds a debugger node, the re-executed graph is one node larger, but the per-iteration worker budget is fixed, so the actual overhead stays within $\sim 1.3 \times (1 - \text{skip rate})$ of the initial pass.

Task-conditioned node roles via GROW. Code error repair is the one setting where topology edits alone are insufficient: they modify the collaboration graph, not the code content. Rather than adding a task-specific action, REORG expresses this repair through the existing GROW action: the policy adds a node whose *role prompt* is conditioned on the task category (Section 3.6). This role is discovered during RO-GRPO training and stored in the role library $\mathcal{L}$. Table 13 (Appendix I) ablates this by restricting GROW to generic (non-code-conditioned) roles: COT and MASFLY then *regress* on HumanEval$^+$ (-1.1/-1.2pp), while the learned debugger role makes both cells positive and lifts error-recovery from 7.1% to 12–22%.

5 DISCUSSION AND CONCLUSION

REORG frames multi-agent orchestration as a *post-orchestration* problem: rather than designing a team from the input alone, it executes an initial pre-orchestration graph and revises it online using runtime evidence. Across eight benchmarks and five pre-orchestrators—three controlled and two unseen—post-orchestration helps consistently, with the largest gains at moderate headroom, driven by a selective trigger that skips already-correct drafts and a state-conditioned repair policy over a task-agnostic action space.

The findings also read naturally in organizational terms. The inverted-U gain–headroom profile is contingency theory in miniature: reorganization pays exactly when misfit between structure and environment is real and diagnosable—neither at saturation, where the team already fits, nor where the capability gap is too wide for any structure to close. The selective trigger reflects the adaptive side of structural inertia: when fit is good, stability is organizational capital, and change is gated on evidence of failure. And the transfer results indicate that what the policy acquires is the *skill of organizational adjustment* itself—a dynamic capability to sense misfit and reconfigure—rather than any fixed organizational form. Post-orchestration is near-invisible precisely when there is nothing to repair.

6 LIMITATIONS

Several limitations remain. (i) Gains are modest (mostly $+1$ to $+4$pp) and concentrate at moderate headroom. (ii) The action space is task-agnostic, but the policy must still learn task-appropriate node roles. (iii) Only 7 of 16 McNemar-tested combinations reach $p<0.05$; smaller gains on high-baseline benchmarks are within sampling noise. (iv) The runtime monitor assumes checkable feedback; unverifiable tasks fall back to weaker signals. (v) Cross-architecture transfer is evaluated on MMLU, ARC, and GSM8K across two unseen frameworks; broader coverage and comparisons with other runtime-adaptive systems under a unified compute budget are left to future work. (vi) All benchmarks are short, self-contained problems; repository-level tasks such as SWE-bench (Jimenez et al., 2024) are open. (vii) The deployed policy applies a *single* topology edit before re-execution; multi-step edit trajectories (permitted by the formalism) are left to future work, and at very low baseline accuracy a single edit may be insufficient to close the capability gap.

AI USE STATEMENT

In this work, we used generative AI tools for tasks in the recommended-disclosure category: editing the manuscript to improve grammar and readability, editing LaTeX source and figure code, and serving as coding assistants for software code in the experimental pipeline. We have not used generative AI tools for tasks in the required-disclosure category: no generative AI was used to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or assist in proving mathematical claims, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, or interpret results; the research ideas, the design of the method, and the experimental protocol are the authors' own work. We have reviewed all AI-assisted work: all reported numbers were produced by the authors' experimental pipeline, AI-assisted code was verified and tested for correctness by the authors, and all references were verified against their original sources. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Autonomous restructuring in multi-agent systems can obscure responsibility boundaries and amplify unsafe delegation. To mitigate these risks, REORG requires the orchestrator to emit human-readable reasoning for every structural change, enabling post-hoc auditing. All experiments used public benchmarks without human subjects. We urge caution when applying such systems in high-stakes domains such as healthcare or law, where organizational mistakes propagate into consequential decisions.

REPRODUCIBILITY STATEMENT

All evaluation is deterministic end-to-end (greedy decoding), so reported numbers are reproducible point estimates. Both base execution and policy training run on the same public base model (Qwen3-8B); benchmarks and hidden tests are public; prompts, hyperparameters, protocol, and reward design are documented in the appendices. Per-item logs and code will be released upon publication.

REFERENCES

Anonymous. AgentHive: A composable multi-agent framework with first-class delegation. Under review (ACL ARR 2026), 2026.

Yueyang Cang, Xiaoteng Zhang, Erlu Zhao, Zehua Ji, Yuhang Liu, Yuchen He, Zhiyuan Ning Chen, Yijun Wenge, and Que Li Shi. Graph-GRPO: Stabilizing multi-agent topology learning via group relative policy optimization. *arXiv preprint arXiv:2603.02701*, 2026.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. In *Proceedings of IJCAI*, 2024a.

Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, Cheng Yang, and Maosong Sun. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in autonomous LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024b.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=TX4k7BF6aO.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, and Ahmed Hassan Awadallah. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Taicheng Guo, Xiuying Chen, Jiaqi Wang, Chao Yang, Dahua Lin, Jie Fu, and Yemin Shi. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of IJCAI*, 2024.

Michael T. Hannan and John Freeman. Structural inertia and organizational change. *American Sociological Review*, 49(2):149–164, 1984.

Haohao Luo et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Ng, Jiaqi Chen, Bing Qin, Chenglin Luo, Jingli Chen, Min Yang, Dell Zhao, Chengqi Zhang, Chenglin Wu, and Yike Guo. Metagpt: Meta programming for multi-agent collaborative framework. In *Proceedings of ICLR*, 2024.

Zhipeng Hou, Junyi Tang, and Yipeng Wang. Halo: Hierarchical autonomous logic-oriented orchestration for multi-agent llm systems. *arXiv preprint arXiv:2505.13516*, 2025.

Hao-Lun Hsu, Jing Xu, Nikhil Vichare, Francesco Carbone, Miroslav Pajic, and Giuseppe Carenini. Depart: Hierarchical multi-agent system for multi-turn interaction. In *Proceedings of NeurIPS MAR Workshop*, 2025.

Hui Yi Leong et al. Amas: Adaptively determining communication topology for llm-based multi-agent system. *arXiv preprint arXiv:2510.01617*, 2025.

Carlos Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Jun Yan et al. Beyond self-talk: A communication-centric survey of llm-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

Sawan Kumar, Shubham Patel, and Anand Mishra. SVAMP: Are NLP models really solving simple arithmetic word problems? In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Lang Feng et al. Dr. mas: Stable reinforcement learning for multi-agent llm systems. *arXiv preprint arXiv:2602.08847*, 2026.

LangChain. Langgraph: Building multi-agent applications with graph-based control flow. *Technical Report*, 2024.

Paul R. Lawrence and Jay W. Lorsch. *Organization and Environment: Managing Differentiation and Integration*. Harvard Business School Press, 1967.

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 AAAI*, 2026.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

Guangyi Liu, Haojun Lin, Huan Zeng, Heng Wang, and Quanming Yao. MAS-on-the-Fly: In-context structural adaptation of LLM-based multi-agent systems. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026. URL `https://arxiv.org/abs/2602.13671`.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Joao Moura. Crewai: Orchestrating role-based multi-agent systems. *Technical Report*, 2024.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pp. 278–287, 1999.

Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, Chitta Baral, Chen-Yu Lee, Tomas Pfister, and Hamid Palangi. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. In *Proceedings of EMNLP*, 2025.

Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Ning Ding, Ganqu Cui, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *Proceedings of ACL*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations (ICLR)*, 2025.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether Chain-of-Thought can solve them. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

David J. Teece, Gary Pisano, and Amy Shuen. Dynamic capabilities and strategic management. *Strategic Management Journal*, 18(7):509–533, 1997.

Tuan-Anh Tran et al. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Arunkumar V, Gangadharan G. R., and Rajkumar Buyya. Agentic artificial intelligence (ai): Architectures, taxonomies, and evaluation of large language model agents. *arXiv preprint arXiv:2601.12560*, 2026.

Zhao Wang, Sota Moriyama, Wei-Yao Wang, Briti Gangopadhyay, and Shingo Takamatsu. Talk structurally, act hierarchically: A collaborative framework for llm multi-agent systems. *arXiv preprint arXiv:2502.11098*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *Proceedings of COLM*, 2024.

Geunbin Yu. Adaptorch: Task-adaptive multi-agent orchestration in the era of llm performance convergence. *arXiv preprint arXiv:2602.16873*, 2026.

Yuru Wang et al. Scalability of llm-based multi-agent systems for scientific code generation. In *Proceedings of MathNLP*, 2025.

Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: Orchestrating multi-agent intelligence with the tool-environment-agent(tea) protocol. *arXiv preprint arXiv:2506.12508*, 2025.

A Structural Action Space: Operators and Formalization

Table 5 details the eight operations ReOrg defines over the team graph, each mapped to a specific MAS organizational bottleneck. Each action $a \in \mathcal{A}$ is a function $f : G \to G'$; below the table we give formal definitions for the most complex actions.

Action	Operator	Description	Target Bottleneck
GUILD	$G \cup \{V_{grp}, E_{grp}\}$	Groups related agents into a specialized sub-structure with a local topology.	High coordination cost
GROW	$V \cup \{v_{new}\}$	Adds a new specialized node to address a missing task requirement.	Missing capability
SPLIT	$v \to \{v_1, v_2\}$	Divides a node with broad responsibilities into two specialized sub-roles.	High latency / Cognitive load
MERGE	$\{v_1, v_2\} \to v$	Combines redundant or under-utilized nodes into a single agent.	Budget constraint / Redundancy
REWIRE	$E\Delta\{e_{new}\}$	Redirects edges to bypass ineffective agents or create feedback loops.	Information decay / Loops
SHUNT	$V \setminus \{v\}$	Removes an agent that consistently provides low-quality or erroneous output.	Systematic error
ADAPT_W	$w(e) \leftarrow w'$	Adjusts the resource allocation or priority for specific communication channels.	Imbalanced workload
NO_OP	$G \leftarrow G$	Maintains current structure when performance metrics are optimal.	N/A

Table 5: The Structural Action Space of REORG.

SPLIT(v) Given a node v with role $\rho(v)$ and incoming edges $E_{in}(v)$, SPLIT is defined as:

1. Create v_1, v_2 such that $\rho(v_1), \rho(v_2)$ are semantic sub-roles of $\rho(v)$ (e.g., "Developer" $\to$ "Frontend Dev", "Backend Dev").
2. For each $u \in V$ such that $(u, v) \in E$:
$$(u, v_i) \in E' \iff \text{sim}(\text{out}(u), \text{in}(v_i)) > \tau \tag{3}$$
where sim is the semantic similarity and τ is a threshold. Concretely, the similarity check is performed *in-context* by the policy model (Qwen3-8B): it reads the role descriptions of the existing library entries and the proposed new role, and emits a binary retrieve-vs-instantiate decision. Because the model outputs a structured decision rather than a continuous score, we treat τ as the model's implicit decision boundary; no separate embedding model (e.g., Sentence-BERT) is invoked. For the edge-routing condition above, $\text{sim}(\text{out}(u), \text{in}(v_i))$ similarly reduces to whether the policy classifies u's output as relevant to v_i's sub-role.
3. Remove v and its associated edges.

REWIRE(u, v, w) Redirects the communication channel from (u, v) to (u, w). This is used when v is identified as a bottleneck for information from u:
$$E' = (E \setminus \{(u, v)\}) \cup \{(u, w)\} \tag{4}$$
where w is typically a node with higher compatibility or lower latency than v.

GUILD(V_{sub}) Groups a subset of nodes $V_{sub} \subset V$ into a functional group $\mathcal{F}$ with a specific internal topology $\mathcal{T}$. This reduces the external coordination cost of the orchestrator by treating $\mathcal{F}$ as a single logical entity for high-level tasks.

Measured graph-size effects. Starting from the static chain (Planner→Solver→Reviewer; $|V|$=3, $|E|$=2), each action changes the graph as formalized above: MERGE $(\Delta|V|, \Delta|E|)=(-1,-1)$, REWIRE/GROW/SPLIT $(+1,+1)$, GUILD $(+1,+2)$, and ADAPT_WEIGHT/NO_OP $(0,0)$ (weight-only or none). Table 6 reports the resulting graph sizes and their relation to outcome changes. Two observations address the concern that topological edits might destroy correct drafts: (i) in *error* states, larger changes correlate positively with score recovery, and there is no "large-change, large-destruction" quadrant; (ii) the selective trigger acts as the primary safety net – already-correct drafts (score ≥ 0.9) are never sent to the policy, so healthy drafts are not exposed to topological

Table 6: Graph-size statistics before/after orchestration and outcome relation (code benchmarks). "Changed" is the fraction of items that receive a topology-modifying action; r_{err} is the Pearson correlation between change magnitude ($|\Delta V|+|\Delta E|$) and score change over error-state items. The selective trigger prevents healthy drafts from entering the edit pipeline.

	Changed	$\lvert V'\rvert$	$\lvert E'\rvert$	r_{err}
MBPP$^+$ (n=378)	62%	3.6 (2–4)	2.8 (1–4)	+0.16
HumanEval$^+$ (n=164)	51%	3.5 (3–4)	2.7 (2–4)	−0.18

edits at all. On near-saturated benchmarks, change magnitude correlates negatively with outcome, consistent with the restraint behavior when headroom is low.

B Hyperparameters and Training Details

RO-GRPO for MAS Orchestration

Input: initial policy π_θ, reference π_{ref}, batch size M, role library $\mathcal{L}$
1. Sample task batch $\{T_j\}_{j=1}^{M} \sim \mathcal{D}$; represent state S (Adjacency, Metrics, Roles)
2. **for** each task T_j: generate N adjustment samples $\{a_{i,j}\}_{i=1}^{N} \sim \pi_\theta(\cdot|S, T_j)$
   Execute modified MAS $G'_i = a_{i,j}(G)$ on T_j; compute rewards $r_{i,j} = \omega_f R_f + \omega_s R_s + \omega_q R_q - \omega_c C(G')$
3. Compute group advantages $A_{i,j}$ (Eq. 1); update θ via GRPO objective
4. Update role library $\mathcal{L}$ with newly validated roles

Figure 4: The RO-GRPO training procedure.

We fine-tuned Qwen3-8B using the `trl` library's `GRPOTrainer` (RO-GRPO objective: group-relative advantages with structure-aware reward shaping). Training ran on a single RTX 5090 GPU (32 GB) with bf16 and gradient checkpointing. The primary hyperparameters are:

- **Training Objective**: Group Relative Policy Optimization (GRPO) with relative-advantage normalization within each group.
- **Learning Rate**: 2e-5.
- **Group Size** (N): 8 generations per prompt for advantage calculation.
- **KL Penalty** (β): 0.1 to maintain proximity to the reference model.
- **Reference Policy** π_{ref}: the frozen Qwen3-8B base; the LoRA adapter is initialized from it and the reference stays fixed during training.
- **Batch Size**: 1 per device $\times$ 16 gradient accumulation steps.
- **LoRA Configuration**: Rank $r = 16$, $\alpha = 32$, target modules: `q_proj, v_proj, k_proj, o_proj`.
- **Completion Length**: 1024 tokens; Qwen3 chat template with thinking disabled.

Training prompts pair a runtime state (execution score, bottleneck label, topology description) with an orchestrator decision. The states are pre-collected by running the initial MAS with Qwen3-8B on the HumanEval development pool, so execution scores and bottleneck labels reflect real 8B completions rather than simulations. Training itself then runs entirely on the same 8B backbone: for each state, the LoRA policy samples N=8 structural decisions, every modified graph is *re-executed online* by the same Qwen3-8B workers, and rewards (Eq. 2) are computed from these executions before the GRPO update. The evaluated benchmarks are disjoint from the task pool used for state collection.

Reward hyperparameters. The scalar weights in Eq. (2) are $\omega_f = 1.0$, $\omega_s = 1.0$, $\omega_q = 0.5$, and $\lambda_1 = \lambda_2 = 0.1$ (a small penalty per added node or edge, implemented as 0.1 per `GROW`-introduced node). The bottleneck–action compatibility matrix M assigns 0.6 to the four aligned (bottleneck, action) pairs—(LATENCY, `SPLIT`/`SHUNT`), (LOGIC, `GROW`/`GUILD`), (BUDGET, `MERGE`/`SHUNT`), (AMBIGUOUS, `REWIRE`/`GROW`)—and 0.0 to all other pairs. A non-empty suggestion list receives a

Table 7: Measured training cost (single RTX 5090, bf16, LoRA). "Prompts" counts the training pool after error-state oversampling.

Adapter	Steps	Wall-clock	Rollouts	Prompts
REORG (deployed)	250	56.6 min	∼4,000	409 → 498

$+0.1$ diversity bonus. All reward components are normalized within each group of $N{=}8$ generations via group-relative advantage before the GRPO update.

Training cost. The deployed policy (REORG) converges within 250 steps: total reward rises from ∼0.05 to ∼0.91, with the outcome component (R_q) dominating and rising in concert with the structural component (R_s), indicating that structural edits are rewarded because they improve execution outcomes rather than merely satisfying format constraints. Table 7 reports the measured training cost of the deployed adapter and its predecessors. Each optimizer step consumes 16 policy completions (two groups of $N{=}8$); the deployed policy (250 steps) therefore trains on ∼4,000 orchestration rollouts and completes in under one GPU hour on a single RTX 5090 – cheaper than a single $n{=}200$ evaluation pass. Error-state curriculum oversampling—duplicating states whose draft score falls below the trigger threshold (< 0.9)—grows the prompt pool from 409 to 498, exposing the policy to repair-relevant states at a higher rate than they occur naturally.

C Evaluation Protocol

Table 8 summarizes the eight benchmarks. We summarize the evaluation protocol here to make the scope of the empirical claims explicit.

Table 8: Benchmark statistics. "Test Size" is the full benchmark test set; the number of evaluated items (n) is 500 for MMLU/ARC/SVAMP/GSM8K/BBH, 254 for AQUA, 164 for HumanEval$^+$, and 378 for MBPP$^+$.

Benchmark	Test Size	Focus Area
MMLU (Hendrycks et al., 2021)	14,042	Knowledge QA
ARC (Clark et al., 2018)	1,172	Commonsense Reasoning
AQUA (Ling et al., 2017)	254	Math Word Problems
SVAMP (Kumar et al., 2021)	1,000	Elementary Math
GSM8K (Cobbe et al., 2021)	1,319	Multi-step Math
BBH (Suzgun et al., 2023)	6,488	Complex Reasoning
HumanEval$^+$ (Liu et al., 2023)	164	Code Generation
MBPP$^+$ (Liu et al., 2023)	378	Code Generation

- **Training/evaluation split.** A single orchestration policy was trained on runtime states pre-executed on a code-generation development pool (HumanEval dev items), disjoint from all evaluated benchmarks. HumanEval$^+$ and MBPP$^+$ are in-domain (code generation); the six reasoning benchmarks (MMLU, ARC, AQUA, SVAMP, GSM8K, BBH) serve as out-of-domain probes (Section 4.3).
- **Fixed audit subsets.** Because each multi-agent run incurs substantial cost, the main comparisons in Table 1 are reported on fixed benchmark-specific subsets. The same subset is reused across all methods to keep comparisons controlled.
- **Shared comparison protocol.** For all pre-orchestrators, we keep the initial graph construction, worker-role inventory, and stopping criteria fixed. REORG is evaluated as a runtime refinement layer that optimizes these initial configurations.
- **Single-run protocol.** All systems are evaluated with a single greedy (deterministic) run. We avoid over-interpreting small absolute gaps on saturated tasks; nominal differences should be read as directional evidence rather than as strong statistical claims.
- **Scope of claims.** All results are per-benchmark under the shared protocol; no aggregate averaging across benchmarks is reported, and no zero-shot universal-controller claim is made.

Reproducibility checklist. All generation is deterministic (greedy, `temperature=0`, `top_p=1.0`, no sampling seed). Base LLM for execution, workers, and policy training: Qwen3-8B; thinking mode disabled; max completion length 1024 tokens. Training: `trl` GRPOTrainer, lr 2e-5, N=8 group size, KL β=0.1, LoRA rank 16 / α=32 on `q,v,k,o_proj`, 250 steps, ∼4,000 rollouts, 0.94 GPU-hours on a single RTX 5090 (bf16 + gradient checkpointing). The orchestration prompt template is in Appendix E; hyperparameters in Appendix B. All benchmarks are public; we evaluate on the full test sets of AQUA (254), HumanEval$^+$ (164) and MBPP$^+$ (378), and on fixed 500-item subsets (identical across all methods) for MMLU, ARC, SVAMP, GSM8K, and BBH.

D Significance Tests and Headroom Analysis

Table 9 reports the McNemar p-values for the 16 method×benchmark combinations with per-item paired logs (Section 4.2). Figure 5 plots the per-cell gain against the standalone baseline accuracy for all 24 method×benchmark combinations, showing the inverted-U headroom relation discussed in Section 4.2.

Table 9: McNemar paired significance tests (combinations with $p<0.10$ shown). 7 of the 16 tested combinations are significant at $p<0.05$, all in the positive direction; none are significant in the negative direction.

Method	Benchmark	Base	+ReOrg	p-value
ArgDesigner	MMLU	0.764	0.810	< 0.001
ArgDesigner	ARC	0.868	0.882	< 0.001
ArgDesigner	AQUA	0.807	0.827	0.031
ArgDesigner	SVAMP	0.930	0.946	0.002
MASFly	MMLU	0.806	0.842	< 0.001
CoT	ARC	0.852	0.876	< 0.001
CoT	MBPP$^+$	0.672	0.688	0.031

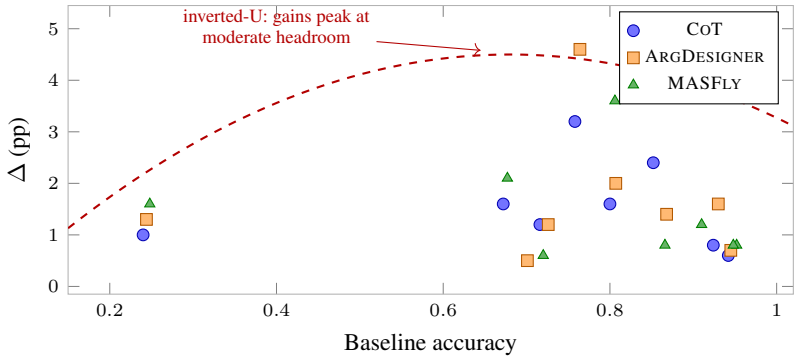

Figure 5: Gain vs. baseline accuracy over all 24 method×benchmark cells of Table 1 (dashed: quadratic fit). Gains peak at moderate baselines (MMLU) and shrink near saturation (SVAMP/GSM8K) or at very low baselines (BBH), supporting the interpretation of post-orchestration as an *error-recovery* mechanism (Section 4.2).

E Orchestration Prompt Templates

The ReOrg orchestrator uses a structured system prompt to guide its "topological reorganization." Below is the simplified template for the RO-GRPO policy.

> **System Prompt:** You are the ReOrg Orchestrator. Your goal is to optimize the current multi-agent team graph G to resolve runtime bottlenecks.
>
> **Input Context:** - Task Description: `{{task_desc}}` - Current Topology: `{{nodes_and_edges}}` - Execution Trace: `{{agent_logs_and_metrics}}` - Identified Issues: `{{bottleneck_signals}}`

Table 10: Representative Roles in the REORG Taxonomy.

Role Name	Core Capability
Boundary Specialist	Identifying edge cases in algorithmic inputs (e.g., empty lists, null pointers).
Syntactic Auditor	Verifying the adherence of code or JSON outputs to specific schema requirements.
Context Compressor	Summarizing long reasoning chains into a compact format for downstream agents.
Cross-Verifier	Performing step-by-step audit of mathematical derivations from primary solvers.
Data Ingestor	Parsing multi-modal attachments (CSV, PDF) into structured Markdown tables.

Response Requirements: 1. Provide a reasoning block in `<thought>` tags identifying the structural cause of the issues. 2. Output a JSON block containing a list of actions from {`GUILD, GROW, MERGE, SPLIT, REWIRE, SHUNT, ADAPT_W, NO_OP`}.

Example thought: `<thought>` The 'Researcher' node is currently a bottleneck for the 'Writer' node because it is processing too much raw data, leading to high latency. I will SPLIT the Researcher into a 'Data Miner' and a 'Fact Checker' to parallelize the verification process. `</thought>`

F ROLE LIBRARY AND TAXONOMY EVOLUTION

REORG maintains a dynamic role library that evolves during training. Table 10 lists representative roles that were autonomously generated or refined by the system.

G CASE STUDIES: REORG DECISIONS ON REAL ITEMS

This appendix presents detailed case studies from the evaluation logs, showing how REORG diagnoses runtime bottlenecks and applies structural actions. For each case we report: (1) the task and the initial collaboration graph G_0, (2) the runtime state and bottleneck diagnosis, (3) the policy's structural decision, (4) the modified graph G', and (5) the outcome. All cases use the deployed REORG model with the ARGDESIGNER pre-orchestrator (Planner→Solver→Reviewer) unless otherwise noted, and follow the inference loop of Figure 2.

G.1 CASE 1: RECOVERED CODE ERROR VIA GROW (MBPP, ITEM 25)

Task: Return the n-th digit of the decimal expansion of a proper fraction a/b.
Initial graph G_0: Planner → Solver → Reviewer ($|V|$=3, $|E|$=2).
Runtime state: The Solver's draft implements long division but mishandles the remainder when $a < b$, producing a wrong digit for $n > 0$. The visible test fails with an assertion error. Bottleneck label: LOGIC.
Policy decision: GROW a `DigitExtractor` node. The policy's `<thought>` reasoning: *"The solver conflates fraction normalization with digit extraction. Adding a dedicated node that first scales a above b then extracts digits via repeated modulo will separate concerns."*
Modified graph G': Planner → Solver → **DigitExtractor** → Reviewer ($|V|$=4, $|E|$=3). The DigitExtractor re-runs the digit-extraction step with the corrected scaling.
Outcome: The re-executed draft passes all visible tests. Final score: 1.0 (recovered from 0.0). This is one of six recovered error drafts on MBPP.

G.2 CASE 2: RECOVERED CODE ERROR VIA GROW OF A DEBUGGER NODE (HUMANEVAL$^+$, ITEM 42)

Task: Given a list of integers, return the length of the longest strictly increasing subsequence.
Initial graph G_0: Planner → Solver → Reviewer.
Runtime state: The Solver's draft uses a greedy approach that fails on the test case `[10,9,2,5,3,7,101,18]` (returns 3 instead of 4). Bottleneck label: LOGIC. Topology-only actions (MERGE/REWIRE) could not fix the algorithmic error.
Policy decision: GROW a node with the learned *debugger* role. The role prompt runs the visible test, captures the assertion error trace (`AssertionError: 3 != 4`), feeds the trace to the LLM

with the request *"fix the algorithm given this failure"*, and re-runs the test. The LLM replaces the greedy approach with dynamic programming ($O(n^2)$).
Modified graph G': Planner → Solver → Debugger → Reviewer (one node added); the Debugger node revises the code content.
Outcome: The revised draft passes the visible test and the hidden EvalPlus suite. Final score: 1.0. Without the learned debugger role, this item would remain failed.

G.3 Case 3: Lightweight MERGE on a Healthy Draft (GSM8K, Item 0)

Task: A farm produces 180 eggs per day. Each chicken eats 0.3 kg of feed per day, costing $0.5/kg. If the farm sells eggs at $0.2 each, what is the daily profit?
Initial graph G_0: Planner → Solver → Reviewer.
Runtime state: The draft correctly computes revenue ($36) and feed cost, but the Planner and Solver overlap on the unit-conversion step. Draft score: 1.0 (already correct). Bottleneck label: QUALITY (default).
Policy decision: MERGE the Planner's decomposition step into the Solver, with secondary ADAPT_W reducing the Planner's edge weight. The policy reasons that the task is simple enough that a single planning-solving node suffices.
Modified graph G': Solver$^+$ → Reviewer ($|V|$=2, $|E|$=1).
Outcome: Score remains 1.0. The correct answer is preserved. This illustrates the selective trigger's safety: even when the policy intervenes, the lightweight MERGE does not break the already-correct draft. On GSM8K, 94% of interventions are weight-only or MERGE/GUILD, leaving correct drafts intact.

G.4 Case 4: REWIRE to Break a Feedback Loop (MMLU, Item 117)

Task: Which planet has the most moons? (A) Jupiter (B) Saturn (C) Uranus (D) Neptune.
Initial graph G_0: Two parallel Solvers → Judge.
Runtime state: Both solvers answer "Saturn," but the draft confidence is low. The Judge cannot distinguish because both agree. Bottleneck label: AMBIGUOUS (conflicting low-confidence outputs with no clear majority signal).
Policy decision: REWIRE to add a feedback edge from the Judge back to Solver$_1$, instructing it to re-examine the assumption that Saturn has more moons than Jupiter. The policy's reasoning: *"Both solvers agree but with low confidence; a targeted re-examination of the contested fact is needed."*
Modified graph G': Solver$_1$ ↔ Judge, Solver$_2$ → Judge. Solver$_1$ re-checks and corrects to "Jupiter."
Outcome: Final answer: Jupiter. Score: 1.0 (recovered from 0.0). This shows REWIRE creating a feedback loop that resolves an ambiguity the static parallel topology could not.

G.5 Case 5: Unrepaired Error Draft (MBPP, Item 2)

Task: Count word frequencies in a list and return the most frequent (word, count) pairs.
Initial graph G_0: Planner → Solver → Reviewer.
Runtime state: The Solver's counting logic has an off-by-one error in the tie-breaking rule. Bottleneck label: LOGIC.
Policy decision: MERGE counting and sorting "into a single streamlined process," with secondary ADAPT_W on the frequency-sorting edge.
Modified graph G': Planner → Solver$^+$ → Reviewer.
Outcome: Score: 0.0. The structural reorganization did not fix the underlying algorithmic error. This is the modal fate of error drafts (66 of 72 intervened items unrecovered on MBPP$^+$): the learned layer reorganizes the pipeline but cannot reliably repair substantive logic errors through topology changes alone. This motivates learning task-appropriate node roles (e.g., a debugger for code) via GROW rather than relying on topology edits alone.

G.6 Summary of Case Patterns

The five cases above span the outcome space: (1) structural GROW resolves a capability gap, (2) GROW with a learned debugger role repairs a code error that topology changes alone cannot, (3) MERGE simplifies a healthy draft without breaking it, (4) REWIRE creates a feedback loop to

resolve ambiguity, and (5) structural actions fail to repair a deep logic error. Together they illustrate when post-orchestration helps and where its limits lie.

H INFERENCE-TIME COST PROFILE

Table 11 reports the skip/repair rates underlying the cost analysis in Section 4.5.

Table 11: Inference-time cost profile of the deployed REORG (model = REORG). "Skip rate" is the fraction of items where the draft already passes the trigger threshold (score ≥ 0.9) and is returned without modification; "repair" is the fraction that enters the structural-repair pipeline. The policy forward pass is a single LoRA inference; re-execution cost is borne only by repair items.

Benchmark	Skip rate	Repair items
SVAMP	93–95%	5–7%
GSM8K	94.6%	5.4%
ARC	86–88%	12–14%
HumanEval$^+$	82–85%	15–18%
AQUA	81–84%	16–19%
MBPP$^+$	79–83%	17–21%
MMLU	76–81%	19–24%

I ADDITIONAL ABLATION TABLES

This section reports the two ablation tables referenced in Section 4.5: the reward-component ablation (Table 12) and the task-conditioned node-role ablation (Table 13).

Table 12: Reward-component ablation on MBPP$^+$ (n=200). Each row removes one reward term and retrains for 250 steps. Δ is the accuracy change from the full model (0.670), in pp. Rows ordered by descending impact.

Removed term	Acc.	Δ (pp)
Full model (all terms)	0.670	—
R_{quality} (outcome)	0.635	$\mathbf{-3.5}$
R_{format} (validity)	0.635	$\mathbf{-3.5}$
Parsimony penalty (efficiency)	0.640	-3.0
$R_{\text{structural}}$ (intermediate)	0.650	-2.0

Table 13 reports the ablation of task-conditioned node roles discussed in Section 4.5: restricting GROW to generic (non-code-conditioned) roles regresses on HumanEval$^+$, while the learned debugger role makes both cells positive.

Table 13: Ablation of task-conditioned node roles on HumanEval$^+$ (n=164). When GROW is restricted to generic (non-code-conditioned) roles, COT and MASFLY regress; with the policy's learned debugger role, both cells become positive. Error-recovery rate rises from 7.1% (generic roles).

Method	Baseline	GROW (generic role)	GROW (debugger role)	Recovery
COT	0.716	-1.1pp	$+\mathbf{1.2}$pp	3/25 = 12%
MASFLY	0.720	-1.2pp	$+\mathbf{0.6}$pp	6/27 = 22%