ARIA: RUNTIME INSPECTION AND AMENDMENT FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

In LLM-based multi-agent systems (MAS), a local mistake can propagate as other agents build on it, turning a correctable error into a failed task. Diagnosing completed runs or repeatedly re-executing them to search for corrections does not provide a learned policy for intervening while execution is still in progress. We present Agent Runtime Inspection and Amendment (ARIA), a framework for learning when and how to repair ongoing multi-agent executions. Our key intuition is to turn execution-verified repairs into reusable supervision, moving repair search into data construction and training so that a learned model can intervene at runtime without repeatedly testing alternatives. First, we collect over 24,000 successful and failed trajectories across four multi-agent frameworks, six benchmarks, and eight host models. From these trajectories, we construct the ARIA dataset, to our knowledge the first large-scale dataset pairing failed multi-agent trajectories with execution-verified repairs. It contains 3,953 verified repair pairs, each specifying what went wrong, where to intervene, and how execution should continue. Then, we develop the ARIA framework for MAS-agnostic runtime inspection and amendment, designed to integrate with any MAS that exposes execution trajectories and supports continuation replacement. Using verified repairs as training examples, we teach a compact model to recognize when intervention is needed and how to correct an ongoing run. The aim is to learn repair strategies that apply to related failures in new configurations, rather than merely reproduce individual corrections. Finally, the trained ARIA model demonstrates strong repair performance in both post-hoc and runtime settings, with generalization to unseen frameworks and host models. Our code is available at `https://anonymous.4open.science/r/ARIA-code`.

1 INTRODUCTION

The use of large language models (LLMs) has expanded from text generation and dialogue (Brown et al., 2020; Touvron et al., 2023) to agents that plan, invoke tools, and adapt to feedback (Yao et al., 2023; Schick et al., 2023; Shinn et al., 2023). Multi-agent systems (MAS) organize agents into coordinated workflows with distinct roles, decomposed tasks, and exchanges through messages or shared artifacts (Li et al., 2023; Wu et al., 2023; Chen et al., 2024; Guo et al., 2024). This division of labor combines specialized capabilities with cross-agent critique to refine intermediate reasoning (Du et al., 2024; Liang et al., 2024). Applications include software development, where MetaGPT and ChatDev coordinate design, implementation, and testing (Hong et al., 2024; Qian et al., 2024); scientific experimentation, where Coscientist integrates planning, search, code execution, and laboratory tools (Boiko et al., 2023); medical reasoning with MedAgents and MDAgents (Tang et al., 2024; Kim et al., 2024); interactive social simulation (Park et al., 2023); and general-purpose web and file tasks with Magentic-One (Fourney et al., 2024). However, distributed work creates dependencies through which an incorrect intermediate result may become the premise for another agent's reasoning or the input to a later tool call (Hong et al., 2024; Guo et al., 2024). Misunderstood messages, inconsistent assumptions, or inadequate verification can carry errors across handoffs, turning a local mistake into an execution-level failure (Cemri et al., 2025; Hammond et al., 2025). These dependencies motivate runtime inspection and repair before actions compound a faulty state, rather than relying solely on final-output evaluation.

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.

Work on execution reliability addresses three questions: what fails, where it originates, and how successful execution can be restored. Failure taxonomies characterize recurring patterns: the Multi-Agent System Failure Taxonomy (MAST) (Cemri et al., 2025) identifies 14 modes spanning system design, inter-agent misalignment, and task verification, while a complementary analysis (Hammond et al., 2025) examines interaction-level risks from miscoordination, conflict, and collusion. Incident-level diagnosis instead locates the decisions responsible for failed runs. The Who&When benchmark (Zhang et al., 2025) evaluates agent- and step-level failure attribution; AgentRx (Barke et al., 2026) uses constraints derived from task requirements and tool specifications to identify critical failure steps; and ErrorProbe (Li et al., 2026) combines backward tracing with execution-based verification and retains verified error patterns. Repair methods then test whether changing suspected failure points improves outcomes. CausalFlow (Bonagiri et al., 2026) searches for minimal counterfactual edits to single-agent traces, whereas DoVer (Ma et al., 2026) tests message or plan changes in multi-agent systems by re-executing the affected continuation. These works support retrospective diagnosis and repair, but runtime intervention must determine from an incomplete trajectory whether a failure has occurred and how to amend the continuation. Learning this capability from past executions presents three connected challenges. First, reliable supervision must pair a failed trajectory with a well-supported diagnosis and an effective correction: task-level labels do not identify the responsible decisions (Zhang et al., 2025; Barke et al., 2026), and plausible repairs require evidence of restored success. Second, collecting such supervision at scale requires selecting productive sources across MAS frameworks, tasks, and backbone models. Configurations that rarely fail supply few examples, yet frequent failures may still yield few verified repairs when correction is difficult; task accuracy alone therefore cannot identify sources rich in useful failure–repair pairs. Third, incident-specific corrections do not directly specify how to act on unseen, unfinished trajectories. A runtime model must decide whether a prefix warrants intervention and generate an amendment that improves subsequent execution. Together, these challenges call for connecting reliable supervision, repair-aware data collection, and joint learning of runtime inspection and amendment.

To address these challenges, we propose Agent Runtime Inspection and Amendment (ARIA), which couples an execution-verified dataset with a framework for learning when and how to intervene in multi-agent executions. We collect over 24,000 successful and failed trajectories across four MAS frameworks, six benchmarks, and eight host models. To obtain useful supervision efficiently, we introduce a source selection rule based on repair optimality that balances failure frequency with repairability, favoring configurations that yield more verified repairs per execution. We connect failure diagnoses to targeted corrections and retain a pair only when the repaired execution passes benchmark evaluation while its unmodified counterpart fails. This process produces the ARIA dataset of 3,953 verified failure–repair pairs, each recording what went wrong, where to intervene, and how execution should continue. Building on this dataset, the ARIA framework trains a compact model in three stages. First, supervised learning teaches the model to recognize when intervention is needed and to generate repairs from verified examples. Second, preference learning (Rafailov et al., 2023) favors successful repairs over failed continuations and diagnoses that help predict repairs for related incidents. Third, policy optimization uses execution feedback on the model's own proposals to improve repair effectiveness and cross-configuration transfer, while refining intervention decisions to recover failures and limit harmful interventions. The resulting model supports both runtime inspection and amendment and post-hoc repair.

Contributions. Our contributions are threefold:

- **A large-scale failure repair dataset.** To the best of our knowledge, ARIA is the first large-scale dataset pairing failed multi-agent trajectories with execution-verified repairs. It spans four MAS frameworks, six benchmarks, and eight backbone models, enabling systematic study of failure repair across diverse multi-agent systems.
- **A novel MAS-agnostic runtime repair framework.** We introduce ARIA, to the best of our knowledge the first MAS-agnostic framework for runtime failure repair. ARIA operates on partial execution trajectories to detect and repair failures without access to the host system's internals, enabling integration into diverse MAS frameworks.
- **State-of-the-art repair performance.** Our post-training approach equips a compact model with both a gate for runtime error detection and a patching capability for failure repair. Extensive experiments show that even a 7B ARIA model achieves state-of-the-art executed repair accuracy in both runtime and post-hoc settings, outperforming zero-shot frontier models.

2 METHODOLOGY

ARIA consists of a corpus and a model trained on it. Section 2.1 determines which source configurations to execute and records all resulting source trajectories, and Section 2.2 describes the three-stage offline procedure that turns those records into a predictor.

2.1 DATASET CREATION

From failure analysis to repair supervision. Multi-agent systems (MAS) exhibit diverse failures, ranging from instruction violations and reasoning errors to inter-agent misalignment and inadequate verification (Cemri et al., 2025; Jia et al., 2026). Existing datasets, such as MAST-Data and Who&When, collect and annotate execution trajectories to characterize failure modes or identify the agents and steps responsible for unsuccessful outcomes (Cemri et al., 2025; Zhang et al., 2025). However, these failure-analysis datasets do not pair failed trajectories with executable repairs and benchmark-verified successful continuations, leaving a gap between understanding failures and learning how to recover from them. While intervention-based debugging has demonstrated that failures can be repaired through targeted modifications and re-execution (Ma et al., 2026), our objective is to turn such recoveries into reusable training supervision. We therefore construct a dataset that pairs failed trajectories with execution-verified repairs. Its construction addresses two connected questions: which source configurations yield the most usable repair examples, and how should repairs be represented so that their effectiveness can be verified rather than merely asserted?

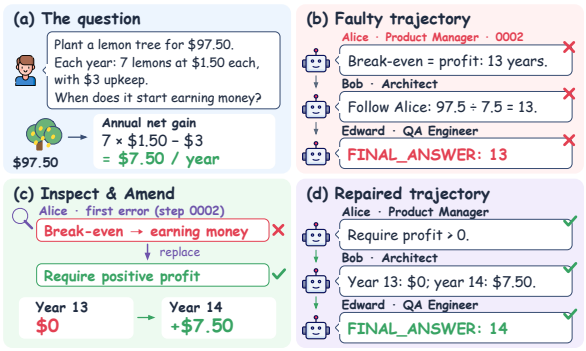


Figure 1: An illustrative ARIA repair: localizing a task misunderstanding and patching the faulty step restores the correct result.

Repair-optimality for source selection. The most useful source of repair supervision is not necessarily the system that fails most often or the one with the strongest backbone; it is the one that produces the most successfully repairable failures per execution. Let a source cell be a triple $c = (A, B, M)$, where A is a multi-agent system with a fixed harness, B is a benchmark with a fixed task distribution, and M is the foundation model that the agents call. Model selection must account for the interaction between these components: the same model can induce different failure distributions under different harness and benchmark combinations, and fault-injection measurements show that stronger foundation models do not uniformly improve robustness and that coordination topology is an equally decisive factor (Jia et al., 2026). We therefore compare candidate models within each fixed pair (A, B), using a fixed repair protocol, rather than ranking them globally. The following principle captures the trade-off underlying this selection.

Definition 1 (Repair-optimality principle)**.** *For a fixed MAS–benchmark pair (A, B), model selection balances two quantities: failure availability, which determines the supply of candidate repair instances, and conditional repairability, which determines how often those failures yield benchmark-verified repairs under a fixed repair protocol. A model that rarely fails provides too few repair instances, whereas a model whose failures are rarely repairable provides little usable supervision.*

Definition 1 states this trade-off qualitatively; the following proposition, proved in Appendix D, makes it operational by identifying the single quantity that governs how much supervision a cell returns per execution.

Proposition 1 (Repair utility score)**.** *For a cell c, let $a(c)$ be the probability that a valid source execution succeeds, and let $r(c)$ be the conditional probability that an attempted repair of a failed source execution succeeds under the benchmark evaluator. Define*

$$S(c) := (1 - a(c))\, r(c). \tag{1}$$

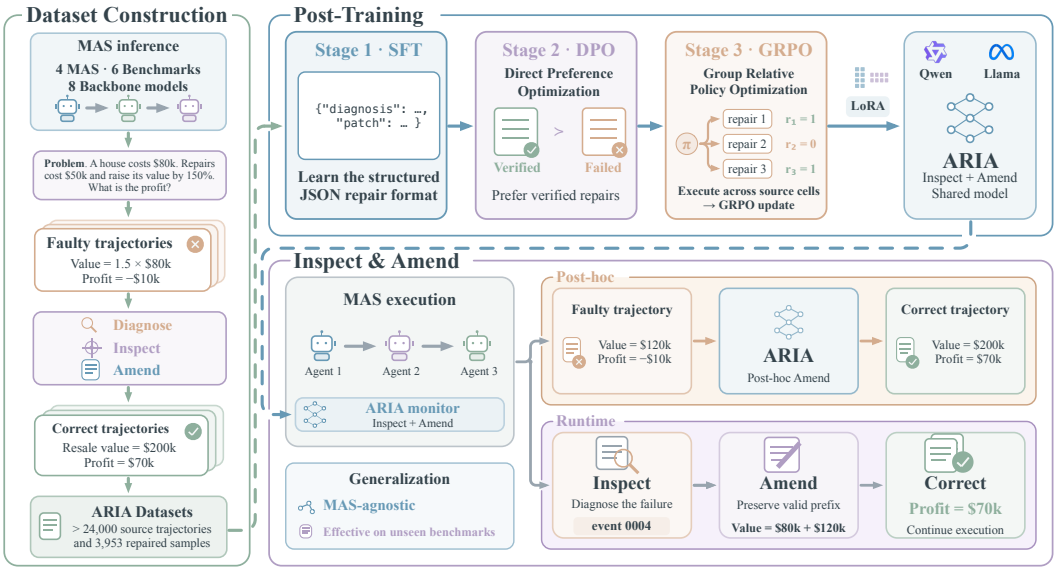


Figure 2: Overview of ARIA, which constructs execution-verified repair data, learns generalizable repair strategies through three-stage post-training, and applies them at runtime to inspect and amend failing multi-agent trajectories.

For N independent valid source executions drawn from the same task distribution, the expected number of successfully repaired failures is $NS(c)$.

Equation 1 thus measures the expected yield of verified repair examples, rather than failure frequency or source-task accuracy alone. Over a finite candidate set $\mathcal{M}$, we select $M(A,B) \in \arg\max_{M\in\mathcal{M}} S(A,B,M)$ independently for each pair in the experimental grid. This rule identifies repair-optimal source cells without imposing a global ordering over models. We estimate $a(c)$ from valid source executions and $r(c)$ as the fraction of attempted repairs that yield a verified success under the fixed repair protocol. Invalid infrastructure runs are excluded from source accuracy, while unresolved repair attempts are reported as a separate status rather than as incorrect task solutions; they do not count as verified repairs. This distinction prevents infrastructure failures from inflating failure availability and unresolved attempts from inflating verified repair yield.

From failure taxonomies to repair attributions. Failure taxonomies describe errors but do not specify how to correct them (Cemri et al., 2025). We introduce *repair attributions* (RA), comprising six groups and 26 categories that link localized failure mechanisms to candidate interventions. Each attribution guides where to cut the trajectory and how to replace its continuation. We retain repair supervision only when paired executions confirm that the patched run succeeds while the unpatched control fails. The attributions then serve as training targets and conditioning information for repair learning. Appendix C provides the full taxonomy, annotation criteria, intervention examples, and validation details.

2.2 Post-Training for Generalizable Repair

We learn structured repairs $y_i = (d_i, k_i, e_i)$ from $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ in three stages: structured supervision, outcome-anchored preference learning, and execution-grounded policy optimization. The first two stages use corpus-derived supervision; the third evaluates current-policy samples to address distribution shift (Ross et al., 2011) and align training with task recovery and intervention costs. Appendix B.5 provides the stage overview and implementation details.

Stage 1: Structured Repair SFT. We first learn the output schema from records with execution-verified repairs. Under constrained decoding, we factorize the supervised loss as

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}\big[\log p_\theta(d_i \mid \tilde{x}_i) + \log p_\theta(k_i \mid \tilde{x}_i, d_i) + \log p_\theta(e_i \mid \tilde{x}_i, d_i, k_i)\big]. \quad (2)$$

Feedback dropout in $\tilde{x}_i$ supports both feedback-conditioned post-hoc repair and feedback-free runtime repair. We jointly train an inspection gate $g_\theta(\tau_{i,\le t})$ to detect whether the certified failure has occurred, using $z_{i,t} = \mathbb{1}[t > k_i]$ for validated cuts and successful repaired prefixes as negatives. Thus, Stage 1 initializes both structured repair and runtime inspection; its checkpoint θ_0 also serves as the Stage 2 reference policy.

Stage 2: Outcome-Anchored Preference Learning. Verified demonstrations do not explicitly rank repairs against failed alternatives. We therefore prefer the verified repair $a^+ = (k_i, e_i)$ to the observed failed continuation $a^- = (k_i, \tau_{i,>k_i})$ under the common context $h = (\tilde{x}_i, \hat{d})$; both retain the prefix $\tau_{i,\le k_i}$. Writing $\Delta_\theta(u \mid h) = \log[p_\theta(u \mid h)/p_{\theta_0}(u \mid h)]$, we minimize the DPO loss (Rafailov et al., 2023):

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\mathbb{E}_{(h,a^+,a^-)} \log \sigma\big(\beta \left[\Delta_\theta(a^+ \mid h) - \Delta_\theta(a^- \mid h)\right]\big), \tag{3}$$

where σ is the logistic sigmoid and β controls reference regularization. This anchors repair preferences to verified outcome contrasts. Diagnoses lack direct outcome labels, so we favor those that improve repair prediction for related training incidents. For a candidate d, we measure its uplift on calibration siblings $F_{\mathrm{cal}}(i) \subset F(i)$:

$$U_i(d) = \frac{1}{|F_{\mathrm{cal}}(i)|} \sum_{j \in F_{\mathrm{cal}}(i)} \left[\log p_{\bar{\theta}}(k_j, e_j \mid x_j, d) - \log p_{\bar{\theta}}(k_j, e_j \mid x_j)\right], \tag{4}$$

where $F(i)$ groups related incidents and $\bar{\theta}$ is a frozen scorer initialized from θ_0. This is a likelihood-based transfer proxy, not executed repair success. High-uplift diagnoses that also improve a held-out training sibling are preferred to low-uplift candidates using the same DPO objective. We then internalize the selected diagnoses alongside verified repairs through supervised learning, yielding θ_1. Family construction, selection criteria, and internalization are detailed in Appendix B.5.2.

Stage 3: Transfer-Grounded Policy Optimization. We initialize π_θ at θ_1 and evaluate its samples through re-execution. For a statically valid record $y = (\hat{d}, \hat{k}, \hat{e})$, the environment of Section 2.1 restores the state at $\hat{k}$, substitutes $\hat{e}$, and resumes execution, returning

$$r(y \mid x) = \mathbb{1}[\text{evaluator accepts}] - \lambda_h \mathbb{1}[\text{harmful}] - \lambda_c\, b(y), \tag{5}$$

where $b(y)$ is execution overhead. Unlike the Stage 2 proxy, this reward measures task recovery and intervention costs through execution. A diagnosis is not executable, so we credit it through the repairs it induces on its source incident and on related incidents from other source cells. For each sampled diagnosis d_q, we generate n source repair records $y_{q,j}$ and n_t sibling repair records $y'_{q,l}$, all conditioned on d_q, and define

$$V(d_q) = (1-\alpha)\frac{1}{n}\sum_{j=1}^{n} r(y_{q,j} \mid \tilde{x}_i) + \alpha \frac{1}{n_t}\sum_{l=1}^{n_t} r(y'_{q,l} \mid \tilde{x}_{\nu_l(i)}). \tag{6}$$

Here $\nu_l(i) \in F(i)$ satisfies $c(\nu_l(i)) \neq c(i)$, where c indexes the framework–benchmark–host-model configuration, and α weights cross-cell transfer. Diagnosis and repair tokens receive separately normalized GRPO advantages (Shao et al., 2024), with regularization toward π_{θ_1}. This favors diagnoses with executed cross-cell utility while optimizing repairs within each diagnosis.

Finally, the inspection gate must balance recovering failed runs against damaging runs that would otherwise succeed. With repair parameters frozen, we optimize complete-task rollouts using the matched-seed reward

$$r_{\mathrm{gate}} = S_{\mathrm{int}} - S_{\mathrm{base}} - \lambda_c\, \Delta\mathrm{EB}, \tag{7}$$

where $S_{\mathrm{int}}, S_{\mathrm{base}} \in \{0, 1\}$ indicate task success with and without gate-controlled intervention under the same seed, and $\Delta\mathrm{EB}$ is the execution-overhead difference. The outcome difference rewards recovered failures and penalizes harmful interventions relative to the untreated baseline. Alternating repair and gate optimization yields θ_2, evaluated in Section 3; sampling, stabilization, and validation protocols are given in Appendix B.5.3.

3 EXPERIMENTS

Our experiments examine ARIA's repair effectiveness, generalization, and the contributions of its post-training stages through three questions. **RQ1:** Can ARIA recover failed executions more effectively and at lower execution cost than existing repair methods in both the post-hoc and runtime intervention settings of Section 2.2? **RQ2:** How well does ARIA generalize to configurations withheld from training, and how does its performance vary with backbone size? **RQ3:** What does each of the three post-training stages contribute to ARIA's performance?

3.1 EXPERIMENTAL SETUP

Backbone models. *Host models* power the executing agents, determine the source failure distribution, and are selected per cell by Equation 1: Llama-3.1-70B-Instruct (Grattafiori et al., 2024), Qwen3-8B/32B, GPT-5.5 (M/H/X reasoning tiers), Claude-Sonnet-4-6, and Claude-Opus-4.8. Only the disjoint *ARIA backbones* are post-trained: Qwen2.5-Instruct at 1.5B, 3B, 7B, and 14B (Qwen et al., 2025) for scaling under a shared pre-training and post-training recipe, and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) as a cross-family control comparable in size to Qwen2.5-7B. Appendix B gives the controls used for backbone comparisons.

Multi-agent systems. We use four coordination structures in the in-domain experiments: MetaGPT's fixed role pipeline (Hong et al., 2024), OpenManus's tool use and delegation (Liang et al., 2025), Magentic-One's ledger-based orchestration (Fourney et al., 2024), and AG2's programmable group chats (Wu et al., 2023). RQ2 additionally treats Magentic-One as an unseen system by retraining ARIA without any of its trajectories. Frameworks run on supported benchmarks; each reported cell identifies its framework, benchmark, and host model.

Benchmarks and evaluation sets. Six benchmarks span four categories: mathematics through GSM-PLUS (Li et al., 2024) and OlympiadBench (He et al., 2024); knowledge through MMLU (Hendrycks et al., 2021); code synthesis and repository repair through HumanEval (Chen et al., 2021) and SWE-bench-Lite (Jimenez et al., 2024); and tool use through GAIA (Mialon et al., 2024). RQ1 uses an *in-domain* test split, while RQ2 compares in-domain and *unseen system* evaluation across backbone scales. Appendix B details data partitioning, held-out-set construction, and fitting restrictions.

Baselines. Post-hoc comparisons include zero-shot repair; the failure-localization methods Who&When (Zhang et al., 2025), AgentRx (Barke et al., 2026), and ErrorProbe (Li et al., 2026); and the intervention-validation method DoVer (Ma et al., 2026). Runtime comparisons include Spark to Fire (Xie et al., 2026), which controls inter-agent messages, and SafeAgent (Liu et al., 2026), an action-mediation method included as a safety-oriented reference. Appendix B gives the baseline inputs, adaptation protocol, model controls, and execution accounting.

Metrics. Success Rate (SR) measures benchmark passes among failed test trajectories in the post-hoc setting or among all test tasks in the runtime setting. Harmful Intervention Rate (HIR) measures interventions that are structurally invalid, violate task constraints, or reduce the graded benchmark score relative to no intervention. Execution Budget (EB) is the average number of full multi-agent executions per evaluated task.

3.2 REPAIR EFFECTIVENESS AND EXECUTION COST (RQ1)

We evaluate ARIA on the in-domain test set in two complementary settings: post-hoc repair of failed trajectories and runtime amendment of ongoing executions.

Post-hoc Repair. In the post-hoc setting, each method receives a complete failed trajectory together with evaluator feedback and task constraints, and the corresponding task evaluator grades the repaired continuation. Table 1 reveals two consistent patterns. First, ARIA achieves the highest overall SR with both backbones and leads on every benchmark, spanning the large mathematics and knowledge pools as well as the smaller code, tool-use, and repository-repair sets. The localization

Table 1: Post-hoc repair performance on in-domain failed test trajectories with evaluator feedback, pooled across four frameworks. We report success rate (SR; higher is better) and execution budget (EB; lower is better), with SR additionally broken down by benchmark. The best value in each comparison block is shown in **bold**.

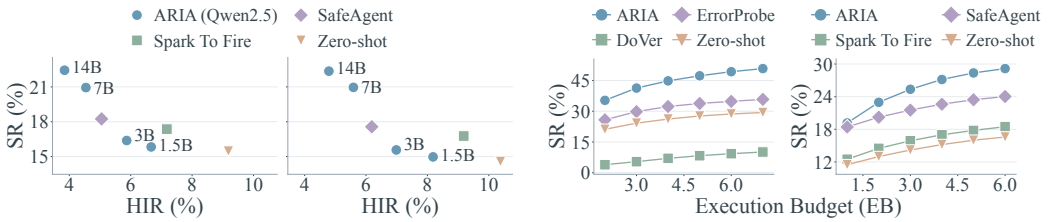
	SR by benchmark ↑						Overall	
Method	GSM-P	MMLU	HEval	Olymp	GAIA	SWE-L	SR ↑	EB ↓
Llama-3.1-8B backbone								
Zero-shot	7.37%	10.5%	2.63%	2.31%	0.00%	1.54%	8.37%	1.98
Who&When (2025)	12.4%	14.7%	0.00%	1.39%	1.92%	3.08%	12.5%	1.98
AgentRx (2026)	8.57%	8.46%	2.63%	3.70%	1.92%	1.54%	7.99%	**1.97**
ErrorProbe (2026)	6.82%	7.42%	2.63%	1.39%	0.00%	0.00%	6.58%	1.98
DoVer (2026)	1.45%	8.36%	13.2%	8.80%	3.85%	3.08%	5.39%	5.75
ARIA	**16.8%**	**18.5%**	**15.8%**	**9.72%**	**7.69%**	**4.62%**	**16.9%**	2.00
Qwen2.5-7B backbone								
Zero-shot	25.1%	21.4%	5.26%	2.31%	1.92%	1.54%	21.2%	1.98
Who&When (2025)	43.5%	26.8%	7.89%	2.78%	3.85%	3.08%	31.6%	1.98
AgentRx (2026)	29.3%	21.9%	2.63%	4.63%	3.85%	0.00%	23.3%	**1.96**
ErrorProbe (2026)	35.4%	22.0%	2.63%	2.78%	1.92%	1.54%	25.8%	1.97
DoVer (2026)	7.00%	4.83%	18.4%	1.85%	5.77%	3.08%	5.69%	3.24
ARIA	**47.0%**	**30.5%**	**21.1%**	**6.94%**	**7.69%**	**6.15%**	**35.3%**	2.00

baselines are strongest on only a subset of reasoning tasks, while DoVer is competitive on HumanEval and OlympiadBench but substantially weaker in aggregate. ARIA's lead also persists from Llama to Qwen, showing that post-training adds value beyond the backbone's stronger zero-shot repair ability. Second, the gain does not come from additional execution. ARIA's average EB remains essentially the same as the zero-shot and localization baselines, whereas DoVer requires more replay to validate interventions through re-execution. In the matched-budget sweep in Figure 3b, ARIA leads at the smallest tested budget and improves steadily; the alternatives remain lower and grow more slowly. Additional budget therefore amplifies an already effective repair policy rather than compensating for weak initial proposals.

Runtime Amendment. In the runtime setting, each method inspects ongoing executions without evaluator feedback. Table 2 shows that ARIA achieves the strongest overall success–harm trade-off with both backbones: its higher aggregate SR is accompanied by markedly lower HIR, rather than by more aggressive intervention. The gains are also consistent across coordination structures, with ARIA leading on OpenManus, MetaGPT, and AG2. The only per-framework exception is Magentic-One with the Llama backbone, yet ARIA still retains the best aggregate SR and lowest HIR in that comparison block. Moving to the stronger Qwen backbone further widens ARIA's aggregate advantage, suggesting that additional capacity benefits both the decision to intervene and the construction of an effective amendment. Because all methods operate under the same single-execution budget, these improvements reflect a better runtime policy rather than replay-based selection. Figure 3a further shows that the 1.5B model already improves over zero-shot amendment and approaches the specialized baselines, while the 7B and 14B models shift decisively toward the upper-left region on

Table 2: Runtime intervention performance on all in-domain test tasks without evaluator feedback. We report per-framework and overall SR, HIR, and EB; HIR is measured against uninterrupted runs. The best value in each comparison block is shown in **bold**.

	SR by framework ↑				Overall		
Method	OpenManus	MetaGPT	Magentic-One	AG2	SR ↑	HIR ↓	EB ↓
Llama-3.1-8B backbone							
Spark to Fire (2026)	17.0%	7.31%	**2.99%**	6.39%	12.5%	7.51%	1.00
SafeAgent (2026)	20.3%	20.6%	0.80%	20.9%	18.4%	5.50%	1.00
ARIA	**20.8%**	**20.8%**	1.20%	**23.7%**	**19.2%**	**1.49%**	1.00
Qwen2.5-7B backbone							
Spark to Fire (2026)	18.6%	8.75%	1.60%	5.70%	13.5%	8.00%	1.00
SafeAgent (2026)	19.6%	18.9%	1.20%	21.8%	17.8%	13.9%	1.00
ARIA	**23.0%**	**23.5%**	**4.59%**	**24.5%**	**21.4%**	**2.00%**	1.00



(a) Runtime success–harm trade-offs are shown for OpenManus (left) and MetaGPT (right).

(b) Success rates under varying EB are shown for post-hoc repair (left) and runtime amendment (right).

Figure 3: ARIA's repair effectiveness and execution cost.

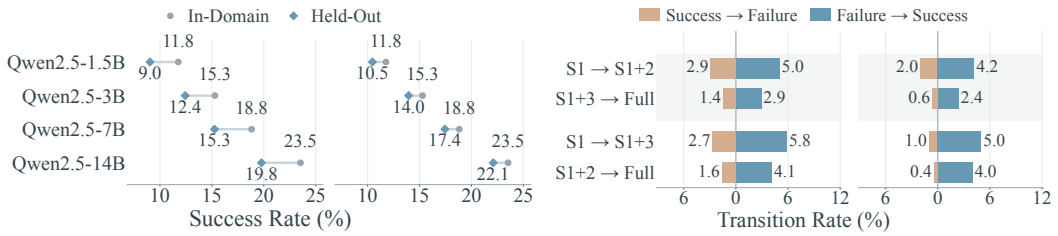
Table 3: Generalization performance across backbone scales in the *in-domain* and *unseen system* (Magentic-One trajectories excluded from training and validation) settings. We report SR and HIR.

		In-domain		Unseen System	
Method	**Backbone**	SR ↑	HIR ↓	SR ↑	HIR ↓
Reference baselines					
Spark to Fire (2026)	Qwen2.5-7B	17.4%	7.18%	16.8%	9.18%
SafeAgent (2026)	Qwen2.5-7B	18.2%	5.05%	17.6%	6.19%
Backbone scale					
ARIA	Qwen2.5-1.5B	15.8%	6.67%	15.0%	8.18%
ARIA	Qwen2.5-3B	16.4%	5.87%	15.6%	6.99%
ARIA	Qwen2.5-7B	20.9%	4.54%	21.0%	5.59%
ARIA	Qwen2.5-14B	22.4%	3.84%	22.4%	4.79%
Cross-family control					
ARIA	Llama-3.1-8B	20.6%	4.81%	20.6%	5.79%

both frameworks, simultaneously raising SR and reducing HIR. Scaling therefore improves gate selectivity as well as repair effectiveness. In Figure 3b, ARIA leads at the lowest budget and remains ahead as the budget grows, confirming that its advantage appears before repeated trials become available and persists when additional execution is allowed.

3.3 Generalization and Scaling (RQ2)

All RQ2 experiments use the runtime setting and the same post-training recipe across Qwen2.5 backbones from 1.5B to 14B. For *unseen system* evaluation, we exclude every Magentic-One trajectory from training, validation, and checkpoint selection, then deploy the resulting model directly on Magentic-One. Success therefore requires transferring both the inspection decision and the amendment strategy rather than recalling system-specific traces. Table 3 shows that unseen-system SR closely tracks in-domain performance at every scale, while HIR increases only modestly. The distribution shift therefore affects intervention confidence more than the model's ability to produce useful repairs. More importantly, the model-size ordering is unchanged: larger backbones monotonically improve both recovery and selectivity, so scaling does not exchange higher success for more aggressive intervention. The small variants remain near the runtime baselines and do not yet dominate them consistently. At 7B, however, ARIA moves beyond the strongest baseline on both metrics, and the 14B model pushes the success–harm frontier further outward. This change suggests that sufficient capacity is especially useful for jointly representing the execution state, deciding whether intervention is warranted, and constructing a compatible continuation. A similarly sized Llama control closely follows Qwen2.5-7B, indicating that the effect transfers across backbone families rather than depending on Qwen-specific behavior. Figure 4a tests the trend under two distinct held-out coordination structures. Every increase in backbone size improves held-out SR, and neither system reverses the in-domain scale ordering. The held-out curves also rise approximately in parallel with their in-domain counterparts, showing that the benefit of additional capacity survives the system shift rather than being confined to familiar trajectories. AG2 remains closer to its in-domain curve, whereas Magentic-One exhibits the larger transfer gap and is consequently the harder setting.



(a) Runtime generalization across Qwen2.5 backbone scales; left: *unseen system* Magentic-One, right: *unseen system* AG2.

(b) Outcome transitions across ARIA post-training variants; left: in-domain tasks, right: *unseen system* Magentic-One tasks.

Figure 4: ARIA's generalization ability and the contributions of its post-training stages.

Table 4: Ablation of ARIA's post-training stages with the Qwen2.5-7B backbone. We report SR and HIR; the best result in each column is shown in **bold**.

	In-domain		Unseen System	
Variant	SR ↑	HIR ↓	SR ↑	HIR ↓
Reference baselines				
Zero-shot	11.7%	10.3%	9.78%	11.4%
ARIA post-training variants				
Stage 1 only (w/o Stage 2 and Stage 3)	14.5%	8.22%	12.4%	8.78%
Stage 1 + 2 (w/o Stage 3)	16.6%	4.15%	14.6%	5.19%
Stage 1 + 3 (w/o Stage 2)	17.6%	3.13%	16.4%	3.79%
ARIA (Stage 1 + 2 + 3)	**19.2%**	**1.49%**	**18.2%**	**2.20%**

3.4 Ablation of the Post-Training Stages (RQ3)

Using Qwen2.5-7B, we examine how the three post-training stages shape runtime repair. As summarized in Table 4, Stage 1 establishes a useful structured repair policy and inspection gate; adding Stage 2 primarily reduces harmful interventions, indicating that preference contrasts sharpen when the model should act, whereas Stage 3 produces the larger recovery gain by grounding proposed repairs in execution outcomes. The full pipeline improves on both partial variants in-domain and on the unseen system, so preference learning and execution optimization remain complementary under distribution shift. Notably, the unseen-system gaps remain directionally similar, suggesting that each stage learns transferable behavior rather than merely fitting framework-specific intervention patterns. Figure 4b confirms that these aggregate gains are not caused by unstable task churn. Every transition to a richer variant converts more failures into successes than successes into failures in both settings. Stage 3 creates the largest positive flow from Stage 1 and remains beneficial after Stage 2, while Stage 2 still contributes net recoveries after Stage 3. Together, the stages progress from structured intervention, to selective intervention, to execution-grounded recovery. Overall, the ablation validates the superiority of ARIA's complete staged post-training design over zero-shot and partial variants in delivering effective, selective, and transferable runtime repair.

4 Conclusion

We introduced ARIA, a MAS-agnostic framework for detecting and repairing failures during multi-agent execution. From over 24,000 trajectories, ARIA constructs 3,953 execution-verified repair pairs that specify the failure, intervention point, and corrected continuation. Its three-stage training combines structured supervision, outcome-anchored preference learning, and execution-grounded policy optimization, moving costly repair search offline. Across diverse frameworks, benchmarks, and host models, ARIA delivers strong post-hoc and runtime repair performance while controlling harmful interventions and execution cost. It further transfers to unseen host models and systems, with performance improving as the repair backbone scales. Ablations confirm that all three training stages contribute to reliable recovery. Together, these results establish execution-verified repair learning as a promising route toward more reliable multi-agent systems.

AI USE STATEMENT

We used generative AI tools solely for language editing and polishing. We did not use generative AI for research ideation, methodology design, code development, data collection, experiments, result analysis, or scientific claims. All AI-assisted text was manually reviewed and verified by the authors. We take full responsibility for the final content of this work, including any text produced with the aid of generative AI.

ETHICS STATEMENT

This work studies automated repair for LLM-based multi-agent systems using established benchmarks and machine-generated execution trajectories. It does not involve human participants, personal or sensitive data, or deployment in high-stakes real-world settings. Nevertheless, automated interventions may introduce errors or unintended behavior and should not be treated as a guarantee of system safety. ARIA should therefore be evaluated under task-specific constraints and appropriate human oversight before real-world deployment.

REFERENCES

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories. arXiv preprint arXiv:2602.02475, 2026. URL https://arxiv.org/abs/2602.02475.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023. URL https://www.nature.com/articles/s41586-023-06792-0.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. CausalFlow: Causal attribution and counterfactual repair for LLM agent failures. arXiv preprint arXiv:2605.25338, 2026. URL https://arxiv.org/abs/2605.25338.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2005.14165.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4082. URL https://doi.org/10.52202/085713-4082. Datasets and Benchmarks Track.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong

Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.10848.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2305.14325.

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, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. arXiv preprint arXiv:2411.04468, 2024. URL https://arxiv.org/abs/2411.04468.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel,

Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2402.01680.

Lewis Hammond, Alan Chan, Jesse Clifton, Jason Hoelscher-Obermaier, Akbir Khan, Euan McLean, Chandler Smith, Wolfram Barfuss, Jakob Foerster, Tomáš Gavenčiak, The Anh Han, Edward Hughes, Vojtěch Kovařík, Jan Kulveit, Joel Z. Leibo, Caspar Oesterheld, Christian Schroeder de Witt, Nisarg Shah, Michael Wellman, Paolo Bova, Theodor Cimpeanu, Carson Ezell, Quentin Feuillade-Montixi, Matija Franklin, Esben Kran, Igor Krawczuk, Max Lamparth,

Niklas Lauffer, Alexander Meinke, Sumeet Motwani, Anka Reuel, Vincent Conitzer, Michael Dennis, Iason Gabriel, Adam Gleave, Gillian Hadfield, Nika Haghtalab, Atoosa Kasirzadeh, Sébastien Krier, Kate Larson, Joel Lehman, David C. Parkes, Georgios Piliouras, and Iyad Rahwan. Multi-agent risks from advanced AI. arXiv preprint arXiv:2502.14143, 2025. URL https://arxiv.org/abs/2502.14143.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2402.14008.

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*, 2021. URL https://arxiv.org/abs/2009.03300.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for A multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.00352.

Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. MAS-FIRE: Fault injection and reliability evaluation for LLM-based multi-agent systems. arXiv preprint arXiv:2602.19843, 2026. URL https://arxiv.org/abs/2602.19843.

Carlos E. 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 *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Yubin Kim, Chanwoo Park, Hyewon Jeong, Yik Siu Chan, Xuhai Xu, Daniel McDuff, Hyeonhoon Lee, Marzyeh Ghassemi, Cynthia Breazeal, and Hae Won Park. MDAgents: An adaptive collaboration of LLMs for medical decision-making. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2404.15155.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2303.17760.

Jiazheng Li, Emine Yilmaz, Bei Chen, and Thu Le. Towards self-improving error diagnosis in multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2063–2077, 2026. doi: 10.18653/v1/2026.findings-acl.98. URL https://aclanthology.org/2026.findings-acl.98/.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2402.19255.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://arxiv.org/abs/2305.19118.

Xinbing Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, Xiao Tang, Bang Liu, Yuyu Luo, and Chenglin Wu. Openmanus: An open-source framework for building general ai agents, 2025. URL https://doi.org/10.5281/zenodo.15186407.

Hailin Liu, Eugene Ilyushin, Jie Ni, and Min Zhu. SafeAgent: A runtime protection architecture for agentic systems. arXiv preprint arXiv:2604.17562, 2026. URL https://arxiv.org/abs/2604.17562.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mrEK16Jy6h.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.12983.

Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 2:1–2:22, 2023. URL https://dblp.org/rec/conf/uist/ParkOCMLB23.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://arxiv.org/abs/2307.07924.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://dblp.org/rec/conf/nips/SchickDDRLHZCS23.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2303.11366.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. MedAgents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://arxiv.org/abs/2311.10537.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn,

Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023. URL https://arxiv.org/abs/2307.09288.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Yizhe Xie, Congcong Zhu, Xinyue Zhang, Tianqing Zhu, Dayong Ye, Minfeng Qi, Huajie Chen, and Wanlei Zhou. From spark to fire: Modeling and mitigating error cascades in LLM-based multi-agent collaboration. arXiv preprint arXiv:2603.04474, 2026. URL https://arxiv.org/abs/2603.04474.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76583–76599, 2025. URL https://proceedings.mlr.press/v267/zhang25cq.html.

A Related Work

Failure taxonomies. A first line of work asks which failures occur in multi-agent executions and how often. Hammond et al. (2025) organize risks from advanced multi-agent systems into three failure modes, namely miscoordination, conflict, and collusion, and seven risk factors that can produce them, and argue that these risks are qualitatively distinct from those posed by individual agents. Restricting attention to LLM-based MAS, Cemri et al. (2025) develop MAST from 150 expert-annotated traces with high inter-annotator agreement, obtaining 14 failure modes grouped into system-design issues, inter-agent misalignment, and task verification, and apply an LLM-as-a-judge annotator to release MAST-Data, a corpus of more than 1,600 traces collected across 7 MAS frameworks. Their case studies show that taxonomy-guided revisions to role specifications and workflows raise task success, which locates the interventions at the level of system design rather than of an individual run. Jia et al. (2026) instead perturb execution directly: MAS-FIRE injects 15 intra-agent and inter-agent fault types through prompt modification, response rewriting, and message-routing manipulation, and reports that stronger foundation models do not uniformly improve robustness and that coordination topology is an equally decisive factor. These studies establish what fails and under which conditions, but a taxonomy label assigned to a completed run does not determine which edit to that run would have recovered the task.

Failure diagnosis. A second line therefore localizes the failure within a specific trajectory. Zhang et al. (2025) formulate automated failure attribution, that is, identifying the agent and the decisive step responsible for a failure, and release Who&When, which contains failure logs from 127 LLM-based multi-agent systems with fine-grained annotations. Their evaluation quantifies how hard the task is: the best of the three methods they study reaches 53.5% accuracy on the responsible agent but only 14.2% on the decisive step, and state-of-the-art reasoning models do not reach practical usability. Barke et al. (2026) target the same localization problem with explicit evidence rather than direct log inspection; AgentRx synthesizes constraints from domain policy and tool schemas, evaluates them step by step, and emits an auditable log of constraint violations from which a judge identifies the critical failure step and category, evaluated on a released benchmark of 115 annotated failed trajectories. Li et al. (2026) propose ErrorProbe, which combines symptom-driven backward tracing, tool-grounded hypothesis validation, and a verified episodic memory. All three operate on a terminated trajectory and return an explanation, so a correct localization narrows the search for a correction without supplying one, and none produces the inspection decision over a partial trajectory that runtime intervention requires.

Failure repair. A third line modifies the execution itself, with objectives that differ in what counts as success. Xie et al. (2026) model collaboration as a directed dependency graph, derive an early-stage criterion for error amplification, and add a genealogy-graph governance layer as a message-layer plugin that suppresses error propagation without altering the collaboration architecture; the objective is to prevent a local error from becoming false consensus rather than to recover a run that has already failed. Liu et al. (2026) share this preventive orientation under a security objective: SafeAgent mediates actions around the agent loop through a stateful decision core and is evaluated on prompt-injection benchmarks, so it blocks unsafe actions rather than restoring task success. The two closest methods repair for the sake of task outcome. CausalFlow assigns each step a causal responsibility score by replacing it and re-executing the downstream computation, then generates minimally edited repairs whose validation yields contrastive pairs of a wrong and a corrected step (Bonagiri et al., 2026); it targets single-agent sequential traces, performs repair at deployment time without parameter updates, and leaves the use of its pairs for preference optimization to future work. DoVer operates on MAS with an explicit orchestrator and sub-agent topology, generates failure hypotheses from logs, and validates them through targeted interventions such as message or plan edits, scoring outcomes by resolution or measurable task progress rather than by attribution accuracy (Ma et al., 2026); it requires framework-level checkpoint and replay infrastructure, and reports that several distinct interventions can independently repair the same failed run. ARIA differs in what it produces. Rather than searching for a correction on each incident at test time, it aggregates execution-verified repairs across MAS frameworks, benchmarks, and foundation models into a single corpus and trains one structured predictor on it, so that a diagnosis, a cut point, and a replacement continuation are emitted in a single forward pass, a prefix-level inspection gate decides on a partial trajectory when to emit them, and no checkpoint or replay hook is required inside the host system. Table 7 in Appendix E summarizes the resulting capability profile.

B Reproducibility and Implementation Details

B.1 Execution Configuration and Benchmark Evaluation

Each experiment is specified by a configuration file that fixes the MAS, benchmark split and task selection, host model, repair model, API protocol, decoding settings, execution limits, and evaluator. We distinguish the source model that generated a failure from the endpoint used for repair generation and paired reruns; these need not be identical. Each sample records both identities and its generation settings.

The four native frameworks use separate dependency environments, while benchmark loading and evaluation use the default environment. The native revisions of OpenManus, MetaGPT, Magentic-One, and AG2 are pinned and recorded with each experiment. Magentic-One additionally requires a Playwright Chromium installation. Repository-local adapters normalize native messages, tool activity, and workspace artifacts without modifying these checkouts.

Table 5: Benchmark verification. Native termination alone does not establish benchmark success.

Benchmark	**Verification**
GSM-Plus	Extracted numerical answer checked against the reference.
MMLU	Extracted option checked against the reference choice.
HumanEval	Generated code executed against the benchmark tests.
OlympiadBench	Answer-bearing mathematics tasks scored with the public mathematical-answer scorer.
GAIA	Validation tasks and attachments; public quasi-exact answer matching.
SWE-bench-Lite	Workspace patch evaluated by the official Docker harness.

The maintained ten-task integration matrix uses OlympiadBench rows 1215–1224, the first ten answer-bearing English, text-only mathematics examples; the initial proof-only rows lack final-answer targets. This diagnostic selection does not define the paper's full task pools. GAIA attachments require an authorized Hugging Face account. Missing official reports, container failures, and unresolved evaluator outputs retain an unknown correctness label rather than becoming wrong answers.

B.2 Source Admission and Incident Construction

A source cell fixes the framework, benchmark, and host-model configuration. Within each source batch, we deduplicate by sample identity and consult the latest result. A cognitive repair incident requires both native completion and an explicit incorrect evaluator verdict. Transport, runtime, tool-backend, and evaluator failures are recorded separately and receive no cognitive label.

The incident contains the public task, constraints, and normalized observable events, each with a stable identifier, role, kind, content, and tool fields. Native traces are preferred; messages provide a fallback when traces contain no observable events. Available workspace artifacts can supply additional events. The default budget is 64 events; longer traces retain the first four and last 60 events. Localization is relative to this visible incident, and cannot certify an omitted event as the earliest error. Post-hoc feedback discloses only that the public evaluator marked the source incorrect. Reference answers, hidden tests, gold patches, and private evaluation details are excluded from model inputs. References remain in the audit record solely for host-side evaluation.

Successful source runs supply non-intervention supervision. For $F > 0$ selected failures and configured rate $\rho > 0$, the default success quota is $\max(1, \lfloor \rho F \rfloor)$, capped by available successful runs; otherwise it is zero. The loader takes eligible successes in source order, unless an explicit count overrides the quota. The configured rate therefore defines a failure-relative quota, not Bernoulli sampling of successful trajectories. The maintained repair configurations set $\rho = 0.25$. These success records invoke no repair generation or paired reruns.

B.3 Localization, Trajectory Cuts, and Repeated Repair Trials

Each trial makes three successive structured predictions: failure localization, repair attribution, and a patch. Localization identifies the first bad event(s), an earlier retained frontier, supporting evidence,

and confidence. Attribution conditions on this localization and specifies one primary RA code, optional secondary codes, responsible roles, a diagnosis, a recovery counterfactual, and a patch operator. The final patch contains only a diagnosis, the retained cut-event identifier, and replacement events. Schema and grounding checks validate event references and their order; they do not establish that a diagnosis is causally correct. The patch must copy the accepted attribution's diagnosis exactly and cannot reproduce the discarded suffix unchanged.

Cut semantics. For an observed trajectory $\tau = (e_1, \ldots, e_T)$, a cut at e_k retains that event and discards the entire suffix after it. The proposed continuation is

$$\tau' = (e_1, \ldots, e_k) \parallel (\hat{e}_1, \ldots, \hat{e}_L). \tag{8}$$

A null cut denotes an empty retained prefix ($k = 0$). The cut must precede the earliest localized bad event. Replacement identifiers must not collide with retained identifiers, and the model cannot fabricate tool-result events. Patches contain at most eight replacement events and 12,000 characters in their canonical JSON serialization.

Independent trials and validation retries. The configuration file sets the number n of complete localization–attribution–patch chains. Each chain starts from the same incident rather than refining a previous trial. Bounded structured-output retries correct a stage's invalid response and are logged within that trial; they are not additional independent trials. The maintained construction configurations currently use $n = 1$; 18 allow two structured-output retries per stage and six allow eight. Increasing n preserves every trial, including invalid generations and failed repairs. Independent calls do not imply distinct outputs under deterministic decoding.

Paired execution and acceptance. The host runs one shared unpatched control and one patched execution per complete trial, in separate workspaces under the declared repair endpoint. The control receives the original task. A patched run receives the task, retained prefix, attribution, and replacement events, with instructions to enact the patch and continue. Both use the benchmark's registered evaluator. A positive repair requires the patched execution to pass and the fresh control to fail; a passing source-independent rerun is insufficient. All complete trials are executed, and the first verified success is selected as the canonical repair. Every verified trial can contribute training data; there is no model vote or ranking over trials.

The native backend reconstructs a continuation context rather than restoring the framework's complete internal state. Paired success therefore validates recovery under this reconstruction protocol, not checkpoint-exact causal replay.

B.4 Training Projection and Data Partitioning

The detailed repair records are the source of truth. Aggregation derives training examples from these records using a shared projection procedure, rather than concatenating potentially stale per-experiment exports. It deduplicates record identifiers, rejects conflicting records, checks answer blindness, and requires each DPO input/chosen pair to match its structured-repair SFT row. Counts of source tasks, repair trials, verified repairs, and training rows are distinct: one incident may yield many gate prefixes and multiple verified patches. Corpus and split manifests retain these counts separately.

Supervision eligibility. Valid model-generated localization and attribution labels from admitted failures can train their respective tasks even when the resulting patch fails. Structured-repair positives and repair preferences require a verified passing patch paired with a failing control; heuristic fallbacks are excluded. For the selected localization with first bad index b, source-prefix gate targets are $z_t = \mathbb{1}[t \geq b]$. If no valid first-bad index is available, only the terminal failure prefix receives a positive label. Success-source prefixes receive zero. Prefixes extending into the selected verified replacement also receive zero; this is a projected training label, not a separate evaluator verdict for every intermediate prefix. Gate inputs always omit evaluator feedback.

Preference negatives. The chosen output is a verified patch. A schema-valid, executed failed trial is used as the rejected output when available. Otherwise, the projector encodes the observed failed

suffix after the chosen cut as a repair record, retaining at most three events and reducing content to meet schema limits. This fallback shares the chosen diagnosis and cut; it is not a newly executed negative. Both outputs use the same event-identifier conventions, and rejection labels remain audit metadata rather than model-visible cues.

Splits and held-out conditions. The splitter assigns task identifiers, not individual prefixes or trials, to nominal 60%/20%/20% train/validation/test partitions using a seeded hash within source-cell strata. A task occurring in several cells receives one global assignment, with its lexicographically first cell defining its stratum. Consequently, per-cell proportions are approximate; strata with fewer than three tasks are assigned to training. All derived rows inherit their task's assignment. Training-view audits additionally check held-out overlap by task identifiers and normalized task text. For unseen-system evaluation, the held-out axis is excluded from training, calibration, and checkpoint selection. Subset caps and task identities are fixed by each study's manifest; rates from different task pools are not directly comparable.

Experimental subsets. RQ1 uses the complete task-level partitions for its in-domain post-hoc and runtime comparisons, whereas RQ2 and RQ3 use separate fixed subsets to make repeated backbone-scaling and ablation runs computationally tractable. In unseen-system conditions, all trajectories from the held-out framework are excluded from training, calibration, validation, and checkpoint selection, with RQ2 holding out Magentic-One or AG2 and RQ3 holding out Magentic-One; the resulting checkpoints are evaluated on the corresponding fixed test subset.

B.5 Post-Training Procedure

The three methodological stages comprise five sequential training phases: structured SFT, repair DPO, diagnosis transfer, repair GRPO, and gate GRPO. The reference trainer uses AdamW and LoRA, with scaling $2r$ for rank r and zero LoRA dropout. Exporting merged weights does not change this into full-parameter fine-tuning. Prompts contain task instructions and canonical JSON. Input reduction preserves event identities and JSON structure; targets are never truncated into incomplete JSON. Oversized or ungrounded records are handled by the configured admission policy and recorded in the report.

B.5.1 Stage 1: Structured Supervision

SFT predicts four target types: gate probability, localization, attribution, and structured repair. Prompt tokens are masked from the causal language model loss. In cascade training, attribution receives the recorded localization and repair receives the recorded attribution; deployment uses the model's own predictions, so errors can propagate between stages. Feedback dropout removes evaluator feedback with the configured probability; it does not paraphrase hidden evaluator information.

B.5.2 Stage 2: Repair Preferences and Diagnosis Transfer

Repair DPO compares the chosen and rejected records under a shared input, using frozen stage-start reference log probabilities. Diagnosis transfer then constructs training-only incident families from sparse token-count representations of feedback and repair edits. Similarity is the mean of their cosine similarities; families are connected components of the thresholded mutual-nearest-neighbor graph. Mining requires at least three family members.

Candidates include the source diagnosis, policy-generated diagnoses, and eligible sibling diagnoses. With model parameters fixed during mining, each candidate is scored by the log-likelihood uplift it gives a sibling's verified repair relative to the same input without that diagnosis. The selected diagnosis maximizes calibration-sibling uplift subject to the configured threshold and positive uplift on one reserved training sibling. The lowest-scoring distinct candidate supplies the preference rejection. The reserved sibling is part of training, not the validation or test set. Diagnosis DPO is followed by SFT on the complete verified repair record with the selected diagnosis substituted. This signal measures likelihood-based transfer, not newly executed recovery.

B.5.3 Stage 3: Execution-Grounded Updates

Repair optimization samples diagnosis branches and multiple patches per branch. In cascade mode, each branch first generates grounded localization and attribution. Valid patches are scored by the configured environment; native environments execute the MAS, whereas evaluator-only and recorded outcome backends do not establish native execution. The default repair reward is

$$r = \mathbb{1}[\text{pass}] - \mathbb{1}[\text{harmful}] - 0.05\, b, \tag{9}$$

where b is the backend-recorded execution cost. Static rejection receives -1 without execution; infrastructure-invalid or unresolved outcomes provide no correctness reward. Diagnosis value combines source and available cross-cell sibling rewards with default transfer weight 0.5. Without a scoreable sibling it falls back to the source reward, which is recorded as missing transfer evidence. Diagnosis/localization outputs receive advantages standardized across branches; patch outputs receive advantages standardized within a branch. Groups without sufficient reward variation are discarded.

The trainer uses a reference-policy penalty (default coefficient 0.02) and a token-level GRPO update. Its current single inner update computes the old log probability by detaching the same forward pass; the probability ratio is therefore one at evaluation. Although clipping bounds are configured, this does not constitute multiple clipped PPO epochs.

Gate optimization evaluates sampled decisions on historical trajectory prefixes, with the first trigger invoking repair. It uses a paired same-host control and the reward $S_{\text{int}} - S_{\text{base}} - 0.05\,\Delta\text{EB}$. Historical-prefix rollouts are distinct from live interception starting at task initialization. The scalar gate mode updates a global logit bias while freezing repair parameters; the contextual mode updates shared LoRA parameters with gate regularization and a repair SFT anchor, and thus does not freeze the repair model. The selected mode is part of checkpoint identity.

Concrete reference recipe. The reference training configuration specifies Qwen2.5-7B-Instruct, seed 42, one epoch, learning rate 2×10^{-4}, LoRA rank 16, gradient accumulation 8, BF16, and gradient checkpointing. It sets a 4,096-token total budget, a 1,024-token prompt budget, feedback dropout 0.25, and DPO $\beta = 0.1$. Its repair GRPO budget is 12 incidents, three diagnosis branches and three patches per branch, one transfer sibling, and four groups per update; gate optimization uses three rollouts, at most four probes, and one outer round. These are reference-recipe settings, not universal settings for all released checkpoints: experiment-specific configuration files and training reports determine the actual row admissions, sampling budgets, and optimizer updates.

B.6 Integration with Native MAS Runtimes

ARIA is attached at the native model-response boundary. Before a generated response is delivered to its framework, the adapter normalizes the actual model request and pending response into an observable prefix. No final evaluator verdict or future event is available. The inspection gate predicts a probability; a value below the configured threshold delivers the original response. A trigger invokes localization, attribution, and structured repair, followed by host-side validation and capability checks. The host, rather than the repair model, decides whether the proposal can be enacted.

The current four native integrations support at most one pending plain-text message substitution per run. An applicable patch cuts immediately before the pending response and supplies a single compatible replacement with the same role and nonempty changed content. Tool calls, protected structured responses, multi-event continuations, and historical rollback cannot be enacted by this interface; such proposals remain shadow records. In shadow mode all proposals are recorded without substitution. The framework continues normally after an applied replacement.

Malformed predictions are recorded as method errors. Under fail-open operation, the original response is delivered, but the error is not relabeled as a valid negative gate decision. Audit records retain the input prefix, raw ARIA calls, proposed patch, original and delivered responses, and enactment status. A proposal or unchanged replacement alone is not evidence of an intervention.

Baseline adaptations and controls. The comparison methods are repository-local adaptations over the common event and execution interfaces, rather than executions of official baseline repositories. Post-hoc methods receive observable failed logs, public feedback, and task constraints. Zero-

shot emits the common repair schema; Who&When, AgentRx, and ErrorProbe add their localization procedures before shared repair generation. DoVer generates interventions for segmented trials and evaluates them through the declared replay backend. Live Spark to Fire and SafeAgent adaptations receive prefixes without feedback and share the pending-text substitution limit; SafeAgent is a text-boundary reference, not its full safety architecture. Within a comparison panel, task identities, host configuration, execution limits, and decoding settings must match. Method-specific prompts and additional calls remain visible in the audit and cost records.

B.7 Success, Harm, and Execution-Cost Accounting

Denominators. Let $\mathcal{P}$ be the fixed evaluation pool. Post-hoc evaluation contains evaluator-confirmed failed source trajectories; runtime evaluation contains all selected tasks, including those that never trigger intervention. For a fully resolved pool of size N, let $s_i \in \{0, 1\}$ denote benchmark success, q_i the graded score, and q_i^0 the paired untreated score. The runtime composite harm indicator is

$$h_i = \mathbb{1}\big[u_i \vee v_i \vee (a_i \wedge q_i < q_i^0)\big], \qquad \mathrm{SR} = \frac{\sum_i s_i}{N}, \quad \mathrm{HIR} = \frac{\sum_i h_i}{N}, \tag{10}$$

where u_i denotes a malformed generated prediction, v_i an explicitly recorded task-constraint violation, and a_i an enacted intervention. With binary grading, outcome harm reduces to a passing control becoming a failure after intervention. Several harmful events on one task count once. HIR is task-normalized, not normalized by gate triggers or by successful controls. The pooled rate weights tasks equally, not framework or benchmark cells equally. For repeated training seeds, the figure aggregator averages the per-seed rates with equal seed weight.

Resolution and live-report scope. The strict evaluation reporter withholds headline SR/HIR until every planned case is resolved; resolved-subset rates are progress diagnostics. The live artifact reporter separately exposes terminal-pool host success and a *harmful-substitution lower bound*: score regressions with an effective delivered substitution, divided by the full planned pool. It reports unresolved cases and decision errors separately. This lower bound excludes malformed-but-unapplied proposals and must not be substituted for the composite HIR in Equation 10. Regressions without applied intervention are reported separately from harmful substitutions.

Execution budget and repeated sampling. Execution budget is the mean sum of recorded full-MAS execution counts per task. Runtime includes the initial execution; post-hoc budget excludes the already acquired failed source, with source-inclusive cost reported separately. Additional method-required reruns count toward cost. ARIA model calls, tokens, and latency are reported separately: a single-run runtime budget does not imply zero inspection overhead. Dataset construction executes all configured trials. In the separate post-hoc budget sweep, the reporter consumes candidates in order and stops at the first evaluator-confirmed success. A runtime sweep cannot select with hidden evaluator verdicts: the implemented multi-candidate reporter uses agreement on public outputs, breaking ties by candidate order. These evaluation selection rules are distinct from construction-trial selection and from diagnosis/patch sampling during GRPO.

B.8 Artifacts and Resume Semantics

Reproduction requires the complete configuration, repository and native revisions, model/tokenizer identities, source and split manifests, checkpoint lineage, and generation settings. Dataset artifacts retain all prompts, raw and parsed structured responses, validation errors, paired outcomes, and per-sample endpoint provenance. Training exports contain canonical inputs and targets without the generation-prompt wrappers. Reports retain corpus checksums, projection drift, admitted/rejected row counts, backend identity, and completed updates; a configured budget is not evidence of completion.

Repair resume skips a sample only when its identifier is present in the formal dataset. Otherwise, partial sample artifacts are removed and all configured trials restart from localization. Each dataset identity has one writer. Native batch resume separately skips completed samples and retries failed samples in clean workspaces. These sample-level guarantees do not imply exact optimizer/RNG restoration from intermediate training checkpoints.

C Repair Attribution Taxonomy

Table 6: Repair attributions comprise six groups and 26 categories. Each tag names a category associated with a candidate patch operator; colors indicate group membership. Group and category names are shortened for presentation.

Group	Categories
RA-1: Understanding & Planning	**RA-1.1** Task misunderstanding · **RA-1.2** Contract misunderstanding · **RA-1.3** Faulty decomposition · **RA-1.4** Faulty role planning
RA-2: Evidence Grounding	**RA-2.1** Insufficient retrieval · **RA-2.2** Misgrounded evidence · **RA-2.3** Extraction/aggregation error · **RA-2.4** Unsupported claim
RA-3: Reasoning & Decisions	**RA-3.1** Invalid inference · **RA-3.2** Computation error · **RA-3.3** Finite evidence as proof · **RA-3.4** Incorrect decision
RA-4: Agent Coordination	**RA-4.1** Lost shared state · **RA-4.2** Ignored correct evidence · **RA-4.3** Failed handoff · **RA-4.4** Non-progress cycle
RA-5: Tools & Implementation	**RA-5.1** Unexecuted tool call · **RA-5.2** Incorrect tool call · **RA-5.3** Unrecovered tool failure · **RA-5.4** Unmaterialized artifact · **RA-5.5** Faulty implementation
RA-6: Verification & Completion	**RA-6.1** Missing verification · **RA-6.2** Inadequate verification · **RA-6.3** Accepted failed check · **RA-6.4** Premature termination · **RA-6.5** Output handling error

Each attribution identifies one primary localized mechanism; secondary codes can record contributors. For example, treating finite search as proof (RA-3.3) suggests replacing the empirical claim with a deductive argument, whereas an ignored failed check (RA-6.3) calls for revising the decision to accept the result. The attribution proposes what to change; paired execution determines whether the resulting patch provides positive repair supervision.

D Repair Utility: Proof and Counting Unit

Proof of Proposition 1. For valid source execution i, let F_i indicate source failure and R_i indicate successful repair under the fixed repair protocol. Then $\mathbb{E}[F_i R_i] = (1 - a(c))r(c)$. Summing over N source executions gives $\mathbb{E}[\sum_i F_i R_i] = N(1 - a(c))r(c) = NS(c)$. □

The counting unit must remain fixed when estimating $r(c)$. For a protocol with n trials and one shared control, define repair success at the incident level as at least one verified passing patch paired with a failing control. The utility then measures incidents yielding usable repairs per valid source execution. It does not count the number of successful trial-level training rows. Trial count, endpoint, and execution limits must be held fixed when comparing source cells; unresolved outcomes are reported separately and never counted as verified repairs.

E Related-Work Capability Comparison

Table 7 supplements Section A with a comparison of the capabilities explicitly reported by each method.

Table 7: Capabilities that each method explicitly reports, where *verified repair* requires re-executing a corrective modification under a task evaluator and a check mark records a reported capability rather than a performance comparison.

Work	Failure taxonomy	Trace corpus	Step attribution	Test-time intervention	Verified repair	Learned predictor
Multi-Agent Risks (2025)	✓					
MAST (2025)	✓	✓				
MAS-FIRE (2026)	✓	✓		✓		
Who&When (2025)		✓	✓			
AgentRx (2026)	✓	✓	✓			
ErrorProbe (2026)			✓			
Spark to Fire (2026)				✓		
SafeAgent (2026)				✓		
CausalFlow (2026)			✓	✓	✓	
DoVer (2026)			✓	✓	✓	
ARIA (proposed)	✓	✓	✓	✓	✓	✓