SMALL CRITICS MAKE STRONG MODELS COUNT: LEARNING TO VERIFY STRONGER MODELS FOR EFFICIENT AUTORESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

AutoResearch agents improve solutions through repeated experiments, but the intertwined demands of decision-making, execution, and analysis make them difficult to build and train. Using strong models throughout is also costly. We study which small-model capability yields the largest system-level gains from training. We introduce **ResearchWeaver**, which decomposes research into an Orchestrator, Worker, and Verifier. This makes research experience usable as role-specific supervision, so a small model can learn one responsibility rather than the entire process. We develop an automated task synthesis pipeline and collect role-structured trajectories on the resulting executable tasks. By training and replacing one role at a time while retaining strong models in the others, we identify where small-model learning helps most: a trained Verifier can influence subsequent research through its diagnoses and suggested next checks. Experiments on diverse benchmarks show that ResearchWeaver consistently improves performance across all three evaluated frontier models by explicitly structuring research into complementary roles. Training small models on role-structured experience further improves performance at the same source-task fraction, and role-specific experiments identify **Verifier training as the highest-gain target**. With supervision derived from only **$\sim$2K source tasks**, training small Verifiers delivers a **10.6-point gain** on average, retaining **92.9%** of the all-strong-model reference performance while reducing strong-model token use by **51.1% on average** across various autoresearch benchmarks. These findings identify Verifier training as a high-leverage path to efficient AutoResearch.

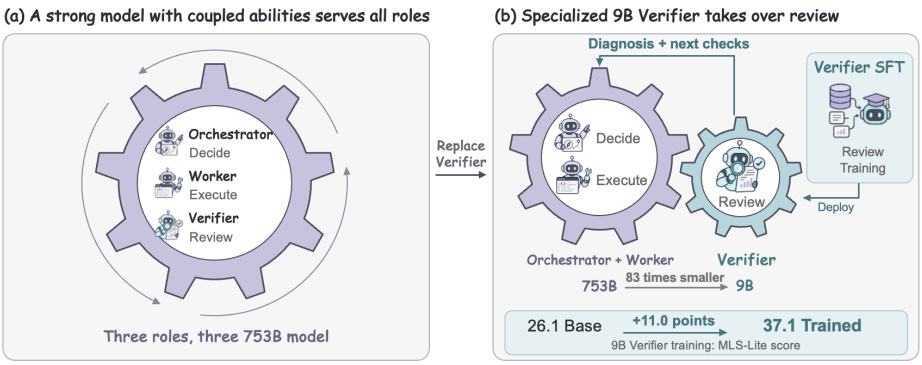
1 INTRODUCTION

Large Language Models (LLMs) can do more than propose a solution: they can test it in an external environment and use the outcome to guide another attempt (Yao et al., 2023; Shinn et al., 2023). This ability matters when a good solution is difficult to specify in advance but can be improved through experimentation. Execution provides evidence about what works, while feedback can help the agent decide what to change next. The opportunity is not simply to generate more candidates, but to make successive attempts more productive.

AutoResearch (AR) brings this goal to research tasks, where agents develop and improve artifacts through repeated experiments (Lu et al., 2024; Lyu et al., 2026; Xu et al., 2026). Building a reliable AR system requires several abilities to work together: choosing a useful experiment, implementing it correctly, and interpreting its results. Relying on a strong model for every step is costly. Training a small model to take over the entire process is also challenging. Research trajectories mix decisions, code changes, and analysis, while the final score reflects their combined effects. Learning from these trajectories therefore requires more than collecting successful runs: it requires turning experience into supervision for the capabilities we want to improve.

Trajectory supervision and heterogeneous model assignment offer ways to improve agent performance and resource use (Du et al., 2026a; Jiang et al., 2026). Capabilities differ in their learning demands and in how much improving them helps subsequent research. This raises a practical ques-

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.



System comparison	(a) All-strong		(b) 9B Verifier	(b) vs. (a)
Strong-model token use	366.1M	→	194.0M	47.0% fewer
AutoResearch performance	41.5	→	37.1	89.4% retained

Figure 1: **A trained small Verifier supports strong-model research with fewer strong-model tokens.** (a) A strong model serves all three roles. (b) A specialized 9B Verifier takes over review, providing diagnoses and next checks while the strong model retains decision-making and execution. Training only the small Verifier raises MLS-Lite performance from 26.1 to 37.1. Compared with the all-strong system in (a), the shown 9B system retains 89.4% of performance while using 47.0% fewer strong-model tokens.

tion: *given experience from the same research tasks, which small-model capability should we train to improve the system most?*

We introduce **ResearchWeaver**, which decomposes AR into three roles: an Orchestrator guides the research direction, a Worker implements and tests ideas, and a Verifier diagnoses problems and recommends next checks. The roles organize research execution and define separate learning targets. We synthesize executable research tasks and collect role-structured trajectories on them. Reconstructing each role's requests, actions, and observations turns these trajectories into focused supervision. Along a research trajectory, reviews of intermediate artifacts and experimental outcomes provide repeated opportunities to learn evidence-based feedback. We then train and replace one role at a time, keeping strong models in the other two.

Experiments on diverse benchmarks show that ResearchWeaver consistently improves performance across all evaluated frontier models: without small-model training, ResearchWeaver scores higher in all 12 backbone–benchmark comparisons (Table 1). Role-structured training trajectories also outperform vanilla trajectories at the same source-task fraction (Table 3). The role comparison then shows that *an untrained small Worker is the strongest replacement, but training the Verifier delivers the largest system-level gains*. This pattern holds for both model families on both training-evaluation benchmarks, and Verifier gains grow across the tested data fractions.

Figure 1 summarizes the resulting deployment: train the small review component and retain strong models for decisions and execution. With supervision from only 2,212 source tasks, the trained small Verifiers average a 10.6-point gain and retrain 92.9% of the all-strong reference performance across different autoresearch benchmarks, reducing strong-model token use by 51.1% on average. Compared with joint training, Verifier-specific training also yields larger gains per successful improvement in the analyzed trajectories, with a similar new-best rate.

Our contributions are threefold:

- **A three-role structure connecting research execution and learning.** ResearchWeaver separates decisions, execution, and review, improving training-free performance across three frontier models and four benchmarks while enabling role-specific data construction and training.
- **A research-data pipeline and evidence for where to train.** We automate task synthesis and role-specific supervision construction. Experiments show the value of role-structured

training data, gains with increasing data, and the largest system-level training gains from Verifier supervision derived from the same research tasks.

- **Small Verifiers for more effective research iterations.** Trained Verifiers retain an average of 92.9% of strong-reference performance and reduce strong-model token use by an average of 51.1%. Compared with Joint-SFT, they yield larger gains per successful improvement at a similar new-best rate, making research iterations more productive.

2 Related Work

Evaluation-driven discovery. Long-horizon AR benchmarks and systems evaluate agents that modify artifacts, run experiments, interpret evidence, and recover from failure (Huang et al., 2024; Lyu et al., 2026; Xu et al., 2026; Si et al., 2026; Ning et al., 2026). Test-time scaling and structured discovery use longer reasoning, repeated exploration, and evaluator-based selection (Muennighoff et al., 2025; Ye et al., 2026); SimpleTES also uses final outcomes for post-training. Our SFT instead targets recorded role behavior and evaluates the recombined system through final task scores.

Harness and process observability. Harnesses determine tool access, context, recovery, memory, and model assignment (Lin et al., 2026; Chen et al., 2026; Zhu et al., 2026b; Zhang et al., 2026). AgentCARD studies heterogeneous roles and API, self-hosted and hybrid deployment costs (Jiang et al., 2026); process analysis localizes differences in framing, implementation, execution, and feedback control (Li et al., 2026a). Our shared interfaces organize execution, data construction, and learning to measure a component's training return within a capable system.

Learning from trajectories and feedback. OpenSeeker and OpenSeeker-v2 construct and filter trajectories for SFT search agents (Du et al., 2026a;b). Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) use textual feedback; process reward models score intermediate steps (Lightman et al., 2024; Qiu et al., 2026; Dihan & Khan, 2026); Generative Verifiers learn verification through next-token prediction (Zhang et al., 2025). Earlier learned verifiers select among generated solutions (Cobbe et al., 2021); our Verifier instead returns diagnostic feedback for subsequent action. ECHO jointly trains a policy and critic using critique-conditioned improvement rewards (Li et al., 2026b). We instead compare offline SFT across AR roles and evaluate a small trained Verifier inside a strong surrounding system. Its feedback is advisory: the Orchestrator must interpret it and decide what to do next.

3 Role-Structured Learning for AutoResearch

We separate history-dependent decisions, bounded implementation, and independent evidence review into explicit interfaces. These organize execution and supervision and enable one-role-at-a-time replacement, without assuming a model size or ease of training for any role.

3.1 The Evaluation-Guided Auto-Research Loop

An AR task x provides an initial artifact y_0, an executable workspace, a budget B, and a reporting objective $J_x(y)$. The agent improves the artifact by inspecting, editing, running experiments and submitting a result. A development evaluator returns $V_x(y) = (r, m)$: an observed score r and metadata m, such as errors or per-setting metrics (Ye et al., 2026). The development score may equal or approximate J_x; improving it need not improve the final reporting objective.

At step t, the agent observes its workspace, interaction history, experimental evidence and remaining budget, then acts:

$$a_t \sim \pi_\theta(\cdot \mid s_t), \qquad s_{t+1} \sim \mathcal{P}_x(\cdot \mid s_t, a_t). \tag{1}$$

One rollout $\tau = (s_0, a_0, \ldots, s_T)$ consists of repeated inspection, modification, and evaluation, and ends at submission, budget exhaustion, or runner termination. The agent uses accumulated experimental evidence to decide whether to refine the current approach, investigate a failure, or explore an alternative. Each action changes the workspace or produces new observations, which inform the next research step. For broader interactive tasks, the evolving object may instead be an environment state or action history, with feedback guiding subsequent actions.

From evaluation to the next research step. An evaluation score measures the current artifact, but does not by itself explain the outcome or determine what to change next (Li et al., 2026a). The next research step requires interpreting the evidence, choosing a direction, and implementing a change. We assign these responsibilities to the Verifier, Orchestrator, and Worker, respectively. These role boundaries connect experimental feedback to subsequent action and define focused learning targets for offline training.

3.2 Three Roles, Three Responsibilities

The **Orchestrator** selects the research direction, delegates work, and decides the next action. The **Worker** implements a bounded request and returns artifacts and observations. The **Verifier** independently reviews those artifacts and experimental claims, diagnoses problems, and proposes next checks. These responsibilities also define the supervision targets.

The evaluator V_x scores the artifact; the Verifier interprets the evidence. Its review should distinguish a flawed idea from an implementation or measurement error, connect explanations to observations, and propose a check that separates alternatives.

For the Verifier, we represent its routed view of the checkpoint and returned feedback as

$$\begin{aligned} s_t^{\mathrm{V}} &= (c_t, \nu_t, o_{\leq t}, b_t), \\ f_t &= (g_t, e_t, u_t, n_t). \end{aligned} \tag{2}$$

The input contains the candidate c_t, version provenance ν_t, observations $o_{\leq t}$, and remaining budget b_t. Feedback contains a diagnosis g_t, supporting evidence e_t, unresolved risk u_t, and a discriminating next check n_t. These fields describe the intended review behavior rather than a mandatory serialization format. Figure 8 in Appendix B details the review responsibilities.

3.3 Connecting Review to the Next Research Step

The Orchestrator dispatches bounded work, requests review at informative checkpoints, and chooses whether to adopt, reject, or investigate the feedback. When requesting a review, it passes the review objective and relevant experimental evidence to the Verifier, which can inspect the workspace and return diagnoses and suggested next steps. The Verifier inspects artifacts and recorded measurements but does not edit, run task tests, allocate budget, or submit. Review can therefore influence later decisions and implementation without taking over their authority. Figure 2 illustrates the resulting Verifier data path and selective deployment; the learning procedure below applies to all three roles.

Dispatch is adaptive, not a fixed three-call pipeline: the Orchestrator can omit either sub-agent or act directly. A configured role replacement is therefore a system intervention, not a guarantee that every task invokes that model. Appendix B.2 formalizes the control flow; the trace analyses distinguish assignment from actual use.

4 Constructing Supervision from Research Experience

We build executable tasks, collect role-structured trajectories, and reconstruct each *role lifecycle*: the history from incoming request to returned response. Retaining decisions and observations aligns supervision with each responsibility. The same runs supply three role-specific datasets and their union for comparing focused and joint training. These datasets let us compare the system-level gains from training each role on the collected research experience. Each task can involve several decisions, implementations, and reviews, so the resulting supervision varies in number and length across roles.

4.1 From Research Sources to Executable Tasks

We map problems, methods, datasets, metrics, and implementations from five years of major ML and NLP conferences; Appendix C.1 illustrates their domain coverage. Sources must provide editable code, a traceable baseline, and reproducible measurement within a practical resource budget.

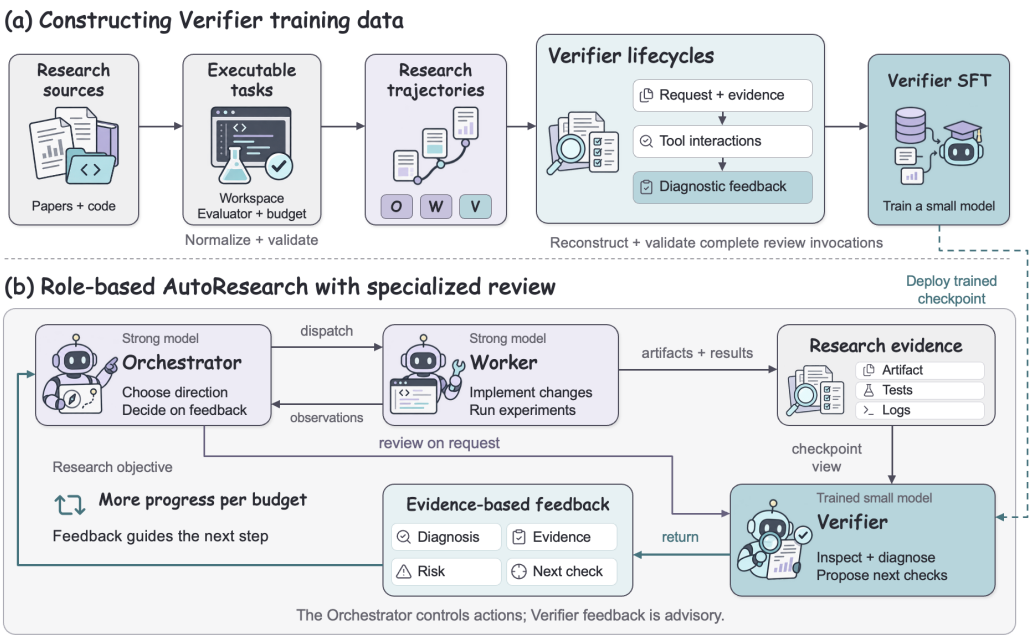


Figure 2: **Verifier data construction and deployment in the role-structured harness.** (a) Research sources become validated executable tasks, whose trajectories provide Verifier lifecycles for supervised fine-tuning. (b) Strong Orchestrator and Worker models retain decisions and execution; the trained small Verifier inspects evidence, diagnoses problems, and proposes checks that can inform subsequent research steps.

Paper prose is normalized into a structured specification containing the intervention, data, metric direction, baselines, code provenance, and resource assumptions:

$$\mathcal{C} \xrightarrow{\text{normalize}} \widetilde{\mathcal{C}} \xrightarrow{\text{deduplicate+validate}} \mathcal{T}. \tag{3}$$

Each accepted task is $x = (I, W, E, B)$: instruction I, editable workspace W, protected evaluator E, and budget B. The evaluator and reference remain outside the workspace. Static, execution, evaluator, and leakage checks establish mechanical eligibility.

Overlap screening. To reduce direct training–evaluation overlap, we screen candidate tasks against held-out benchmarks using task names and file-content hashes (MD5). This removes 147 of 2,359 candidates, leaving 2,212 source tasks whose trajectories provide the training data. The checks target name and byte-level matches, not semantic similarity; Appendix C gives the schema and audit.

4.2 Extract, Train, and Replace One Role

Lifecycle reconstruction. Executing $\mathcal{T}$ with the role-structured harness produces trajectories $\mathcal{R}$. For role $r \in \{\mathrm{O}, \mathrm{W}, \mathrm{V}\}$, reconstruction and filtering yield

$$\mathcal{D}_r^{\text{raw}} = \text{Extract}_r(\mathcal{R}) \xrightarrow{\text{validate+tokenize}} \mathcal{D}_r^{(L)}. \tag{4}$$

For example, a Verifier lifecycle $z_i^{\mathrm{V}} = (s_i, q_i, \tau_i^{\text{tool}}, f_i)$ includes the invocation evidence, request, ordered tool interactions, and returned feedback. A Worker execution appears as a returned observation in the Orchestrator history and as its own Worker lifecycle. This preserves each role's perspective without combining their responsibilities into one training target.

Validation pairs calls with responses, requires a terminal role response, preserves compaction provenance, and checks post-template length. Missing source returns carry explicit markers where needed, so a complete serialized lifecycle need not contain every original observation. Appendix C.2 documents these recovery and filtering rules.

Training and reinsertion. The final datasets are constructed from preserved source sessions with a 512K-token sequence limit, not from a truncated derivative. The final training datasets contain 1,900 Orchestrator, 4,700 Worker, and 7,300 Verifier lifecycles, for 13,900 examples in their natural joint union. The 7,300-example Verifier corpus trains Qwen3.5-9B and Gemma-4-12B; the other role datasets and joint union provide the controlled training comparisons. Multiple lifecycles from one source run are additional supervision units, not statistically independent tasks. Repeated review invocations along a trajectory account for the larger Verifier corpus.

We train on assistant tokens within each role's lifecycles; Joint-SFT uses their union. Each checkpoint is reinserted at its designated role interface and evaluated through system-level task scores, not response-level rewards. Appendix C.3 specifies the objective and optimization settings; Appendix C.2 distinguishes historical training permissions from the runtime contract.

5 EXPERIMENTS

We test which role benefits most from training, whether role-structured experience improves learning, and how supervision choice and data scale affect performance.

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate MLS-BENCH-Lite (Lyu et al., 2026), AutoLab (Xu et al., 2026), PostTrainBench (Rank et al., 2026), and EdgeBench (Zhu et al., 2026a).

Together, these benchmarks span diverse research domains and feedback interfaces. MLS-Lite covers 12 research areas, including robotics, language modeling, vision, reinforcement learning, and scientific applications. AutoLab adds systems, algorithms, model development, and CUDA kernel optimization. PostTrainBench spans mathematics, writing, tool use, science, health, and code through model post-training; EdgeBench extends coverage to formal proofs, professional work, and interactive games and simulators. This breadth tests whether the role structure transfers across different objectives and feedback interfaces, while controlled role-training comparisons use MLS-Lite and AutoLab. Appendix D.2 gives the domain distributions and task counts.

Models. The strong-model backbones are Opus 4.6 (Anthropic, 2026b), GLM-5.2 (Z.ai, 2026), and DeepSeek-V4.1-Flash (DeepSeek-AI, 2026). Local Qwen3.5-9B (Qwen Team, 2026) and Gemma-4-12B (Gemma Team, 2026) use 524,288-token (512K) contexts.

Controls and reporting. We replace one GLM-5.2 role with a Base or trained small model, fixing the other roles, instructions, tools, and configured budgets. We follow each benchmark's official time budgets and resource configurations; for EdgeBench, we report results under a two-hour budget per task. ResearchWeaver uses automatic context summarization to support long research

Table 1: **Baseline versus role-structured execution across four benchmarks.** Each pair uses the same backbone throughout. Scores are percentages (↑); Avg. averages the four benchmarks. †: directly quoted official benchmark results (Lyu et al., 2026; Xu et al., 2026; Rank et al., 2026). ±: standard deviation over three runs.

System	MLS-Lite	AutoLab	PostTrain	EdgeBench	Avg.
Opus 4.6					
Baseline	$35.9^{\dagger}$	$68.0^{\dagger}$	$23.2^{\dagger}$	28.6	38.9
+ Role-structured	40.2 ± 1.2	71.0 ± 0.2	25.1 ± 0.8	30.6 ± 0.4	**41.7**
GLM-5.2					
Baseline	$40.4^{\dagger}$	68.5	31.7	27.5	42.0
+ Role-structured	41.5 ± 0.3	70.5 ± 0.5	33.3 ± 0.6	29.2 ± 0.8	**43.6**
DeepSeek-V4.1-Flash					
Baseline	32.3	73.1	32.7	31.5	42.4
+ Role-structured	35.1 ± 0.5	74.9 ± 0.4	33.9 ± 0.5	33.4 ± 0.2	**44.3**

Table 2: **Verifier training yields the largest gains across both models and benchmarks.** Only the indicated role uses the small model; the others use GLM-5.2. Reference rows use one model throughout (—: no training comparison). Bold: best trained score (↑).

Role replaced	MLS-Lite ↑		AutoLab ↑	
	Untrained	Role-trained	Untrained	Role-trained
Qwen3.5-9B				
Role-structured	9.1	—	18.6	—
Orchestrator	14.5	19.7	27.1	32.3
Worker	32.3	32.9	59.6	61.0
Verifier	26.1	**37.1**	51.1	**64.3**
Gemma-4-12B				
Role-structured	16.2	—	24.3	—
Orchestrator	19.4	24.1	30.8	35.8
Worker	33.1	34.5	63.0	64.6
Verifier	27.9	**38.4**	61.8	**69.3**
GLM-5.2 references				
GLM-5.2 baseline	40.4	—	68.5	—
Role-structured	41.5	—	70.5	—

sessions. Appendix D.3 specifies the compaction trigger and retained history. We report official baselines where available; otherwise, results average three runs under the same settings. We also report research-progress and inference-token statistics. Appendix D details the environments, budgets, and accounting.

5.2 Does Role-Structured Execution Work Across Settings?

Table 1 tests role organization without training, using the same backbone for baseline and role-structured execution. ResearchWeaver improves all 12 model–benchmark scores by 1.1–4.3 points across three frontier models and four benchmarks.

The benchmarks span scientific modeling, robotics, systems optimization, post-training, proofs, and interactive environments, testing diverse objectives and feedback interfaces. Consistent benchmark-level gains support the design's generality across these settings; Appendix D.2 details their domain coverage. Role decomposition thus provides an effective training-free structure as well as interfaces for specialized learning. We next test which role benefits most from training.

5.3 Which Role Benefits Most from Small-Model Training?

We extract each role's supervision from trajectories on the same 2,212 tasks (Section 4.2). Table 2 replaces one GLM-5.2 role at a time: Base scores measure replacement sensitivity, and Base-to-SFT gains measure training benefit. Endpoint scores use the full task set of each benchmark.

Replacement ease differs from training payoff. On MLS-Lite, Qwen Worker starts strongest but gains only 0.6 points (32.3 to 32.9). Verifier gains 11.0 (26.1 to 37.1), while Orchestrator remains weakest (14.5 to 19.7).

Gemma gains 4.7/1.4/10.5 points for Orchestrator/Worker/Verifier on MLS-Lite. AutoLab gains are 7.5–13.2 for Verifier, 5.0–5.2 for Orchestrator, and 1.4–1.6 for Worker, again favoring Verifier training. Archived MLS-Lite runs gain 30.7 points in optimization and 15.3 in time series, versus 0.8 in language modeling. All four AutoLab domains improve: gains are 23.1 for CUDA kernels, 14.6 for model development, 7.1 for puzzles, and 3.2 for systems optimization. The benefit spans different research settings, although its magnitude varies substantially. These differences suggest that review may help more when experimental evidence points to actionable revisions. Appendix G.1 details the statistics and analysis.

5.4 Do Role-Structured Trajectories Improve Learning?

Table 3 separates the effects of role-structured inference and training experience in an all-Qwen3.5-9B system. Each collection protocol uses 25% of source tasks, not equal training-token counts. Starting from vanilla training data and Claude Code (Anthropic, 2026a), we change inference first, then training data.

Table 3: **Role structure improves inference and learning.** Qwen3.5-9B is used throughout. Training comparisons use the same source-task fraction. The untrained row is an archived reference, not a paired training control.

Training data	Inference	MLS-Lite ↑
None (untrained reference)	Role-structured	9.1
Vanilla trajectories	Claude Code	12.1
Vanilla trajectories	Role-structured	13.6
Role-structured trajectories	Role-structured	**15.6**

Changing inference alone improves the same checkpoint from 12.1 to 13.6. Using role-structured training experience adds another 2.0 points, reaching 15.6 under the same inference protocol and source-task fraction. Role structure thus improves both execution and learning under these recipes.

5.5 Does Training on Other Roles Help the Verifier?

Does learning to decide and execute also help a small model review research? We compare Verifier-only supervision with joint training on all three roles. Only the Qwen Verifier checkpoint changes; Orchestrator and Worker remain GLM-5.2. Verifier-SFT uses review data; Joint-SFT adds Orchestrator and Worker data but is still deployed only as Verifier. Both train for two epochs, with different token budgets. Final research scores, rather than standalone review quality, measure how each training recipe supports the downstream research process.
Both improve over Base, but Verifier-only training leads Joint-SFT by 8.8 points on MLS-Lite and 5.9 on AutoLab (Figure 3).

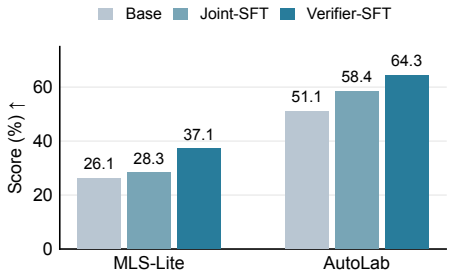


Figure 3: **Verifier-only training leads the tested Joint-SFT recipe.** Only the Qwen Verifier checkpoint changes.

5.6 How Does Performance Scale with Role-Specific Data?

We next vary each role's training-data fraction, fixing the other roles to GLM-5.2 and measuring gains over its Base replacement (Figure 4). The full corpora share a source-task pool; each curve varies the fraction of its role-specific examples. Thus these intermediate fractions describe within-role data scaling, not identical subsets of source tasks across roles.
Verifier gains increase across the reported data fractions, from 2.3 points at 10% to 11.0 at full data. Orchestrator also benefits from more data, reaching a 5.2-point gain, whereas Worker remains near or below its Base performance and reaches only +0.6 at full data. Verifier already gains 4.8 points at 25% and 7.3 at 50%, showing that its improvement develops across intermediate checkpoints rather than appearing only at full data. Appendix C.3 gives the training settings and data-scale results.

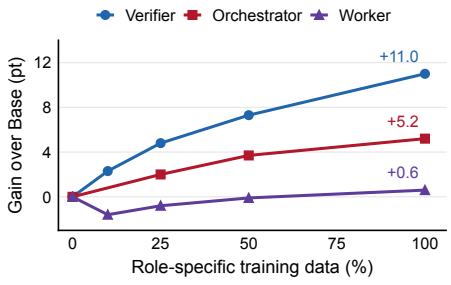


Figure 4: **Verifier gains grow with data.** MLS-Lite gains over Base; fractions use each role's own data scale.

6 Discussion: Choosing High-Leverage Learning Targets

6.1 Observed Improvement Rates and Step Sizes

We compare improvement frequency and size by changing only the Qwen Verifier checkpoint from Joint-SFT to Verifier-SFT, keeping the other roles at GLM-5.2. A new best exceeds all previously recorded scores. Budget-end best observed scores and new-best rates use all 30 MLS-Lite tasks; mean gain uses the 19 where both conditions improve. Gain averages each task's post-initial improvement per subsequent new best, whereas the new-best rate includes the initial positive score.

Larger average gains per successful improvement. Joint-SFT and Verifier-SFT have almost identical new-best rates (58.8% versus 58.9%), but Verifier-SFT has larger mean gains per new best (9.5 versus 13.2 points). The mean best observed score within the recorded budget is also higher (33.2 versus 39.7), distinct from the official final-submission scores. All paired confidence intervals include zero; Appendix E.1 reports uncertainty.

6.2 Inference Allocation: Fewer Strong-Model Tokens

We compare all-GLM-5.2 with Qwen Verifier-SFT, keeping decisions and execution at GLM-5.2. We report performance and inference token usage on MLS-Lite.

With Qwen Verifier-SFT, GLM-5.2 consumption falls by 47.0%, from 366.1M to 194.0M tokens, with 16.8M input/output tokens handled by the small model. On the full MLS-Lite benchmark, Verifier-SFT retains 89.4% of the all-strong-model system's score (37.1 versus 41.5), a 4.4-point decrease. Figure 1 illustrates this allocation; model-size labels denote total parameters, not active compute. Appendix E.2 reports strong- and small-model token usage across role replacements. Gemma Verifier-SFT uses 164.4M GLM-5.2 tokens, a 55.1% reduction, plus 35.4M small-model tokens.

6.3 Replacement Ease and Training Return Are Different

Which role to train. Table 2 compares Base and trained replacements while retaining GLM-5.2 in the other two roles. Worker gives the strongest Base scores in all four model–benchmark comparisons, but training adds only 0.6–1.6 points. Verifier training adds 7.5–13.2 points, the largest gain in every comparison. Choosing a role solely by its untrained score would favor Worker and miss the larger gains available from training Verifier. Deployment suitability and training priority are therefore separate decisions.

Which supervision to use. Figure 3 keeps the deployment fixed and compares two Qwen Verifier training recipes. Verifier-only training exceeds Joint-SFT by 8.8 points on MLS-Lite and 5.9 on AutoLab. For these experiments, training the review role on its own experience is more effective than adding the other roles' data. With the deployment role fixed, broader supervision is not automatically better supervision. The comparison motivates matching training content to the model's intended responsibility, rather than assuming that more varied trajectories necessarily improve that role. The practical choice is therefore both where to train and what supervision to use.

7 Conclusion

This work asks which capability a small model should learn to improve a complex, costly AutoResearch system without mastering the entire research process. ResearchWeaver separates decisions, execution, and review, while our task-synthesis and trajectory-processing pipeline turns research experience into role-specific supervision. Across two model families and two benchmarks, Verifier training yields the largest system-level gains among the three roles. For Qwen, Verifier-specific training also outperforms the tested joint-training recipe. Trained Verifiers preserve much of the all-strong system's performance with fewer strong-model tokens; compared with Joint-SFT, their trajectories show larger gains per successful improvement. The central conclusion is that learning a focused review capability can improve research progress while reserving strong models for decisions and execution.

AI USE STATEMENT

Generative AI systems were used in two roles in this work. First, as components of the experimental systems, generative models performed orchestration, implementation, and verification in the AutoResearch process and generated research trajectories for role-specific supervised fine-tuning. Second, generative AI tools assisted with code debugging, data analysis, and language editing. All AI-assisted outputs were reviewed and validated by the authors. The authors take responsibility for the final text, claims, code, figures, and associated artifacts.

ETHICS STATEMENT

Our research was conducted in accordance with the ICLR Code of Ethics. It involved neither human subjects nor animal testing. All data and source materials, including MLS-Bench-Lite, AutoLab, PostTrainBench, EdgeBench, and the research sources used to build our tasks were obtained and handled according to applicable usage policies, with privacy protected. We made deliberate efforts to minimize bias and discriminatory outcomes throughout the study. No personally identifiable information was collected or used, and no experiment posed privacy or security concerns. We are committed to transparency and integrity in all aspects of this work.

REPRODUCIBILITY STATEMENT

The main text distinguishes benchmark endpoints from matched process analyses and configuration-level from causal evidence. Appendices C and D document data construction, training, and evaluation. Appendices E–G provide supplementary statistics, diagnostic protocols, and trace cases. Each supplementary analysis specifies its controls, measurement definition, and statistical scope.

REFERENCES

Anthropic. Claude Code: Overview. `https://code.claude.com/docs/en/overview`, 2026a.

Anthropic. Claude Opus 4.6. `https://www.anthropic.com/news/claude-opus-4-6`, 2026b.

Zhengyu Chen et al. Co-Harness: Co-evolving harnesses and model weights for LLM agents. *arXiv preprint arXiv:2607.22688*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023. URL `https://arxiv.org/abs/2307.08691`.

DeepSeek-AI. DeepSeek V4.1 Flash: Stronger, faster, more accessible. `https://deepseek.com/news/deepseek-v4-1-flash/`, 2026.

Mahir Labib Dihan and Md Ashrafur Rahman Khan. SWE-Shepherd: Advancing PRMs for reinforcing code agents. *arXiv preprint arXiv:2604.10493*, 2026.

Yuwen Du et al. Openseeker: Democratizing frontier search agents by fully open-sourcing training data. *arXiv preprint arXiv:2603.15594*, 2026a.

Yuwen Du et al. Openseeker-v2: Pushing the limits of search agents with informative and high-difficulty trajectories. *arXiv preprint arXiv:2605.04036*, 2026b.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20271–20309. PMLR, 2024. URL https://proceedings.mlr.press/v235/huang24y.html.

Yinsicheng Jiang et al. Specialize roles, mix deployments: Pushing the cost-accuracy frontier of LLM agent teams. *arXiv preprint arXiv:2606.20629*, 2026.

Yiwei Li, Wanli Yang, Hexiang Tan, Xiangzhou Huang, Zhengyu Chen, Ziran Li, Borun Chen, Shanglin Lei, Huaisheng Zhu, Hao Tian, Fei Sun, Xunliang Cai, and Jingang Wang. Beyond final scores: A systematic evaluation of agents for long-horizon AI research and development. *arXiv preprint arXiv:2608.13417*, 2026a.

Zhicong Li, Lingjie Jiang, Yulan Hu, Xingchen Zeng, Yixia Li, Xiangwen Zhang, Guanhua Chen, Zheng Pan, Xin Li, and Yong Liu. No more stale feedback: Co-evolving critics for open-world agent learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12643–12660. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.576. URL https://aclanthology.org/2026.acl-long.576/.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, et al. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI. *arXiv preprint arXiv:2605.08678*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Fei-Fei Li, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Jingjie Ning, Xiaochuan Li, Ji Zeng, Hao Kang, and Chenyan Xiong. Auto research with specialist agents develops effective and non-trivial training recipes. *arXiv preprint arXiv:2605.05724*, 2026.

Zhisong Qiu et al. Rewarding the scientific process: Process-level reward modeling for agentic data analysis. *arXiv preprint arXiv:2604.24198*, 2026.

Qwen Team. Qwen3.5-9B model card. https://huggingface.co/Qwen/Qwen3.5-9B, 2026.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020. doi: 10.1109/SC41405.2020.00024. URL https://arxiv.org/abs/1910.02054.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training? *arXiv preprint arXiv:2603.08640*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel Candès, Diyi Yang, and Tatsunori Hashimoto. Towards execution-grounded automated AI research. *arXiv preprint arXiv:2601.14525*, 2026.

Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, Shizhe Diao, Jiaxin Pei, Jinsung Yoon, Hao Zhang, Mengdi Wang, Radha Poovendran, Misha Sra, Alex Pentland, and Zichen Chen. Autolab: Can frontier models solve long-horizon auto research and engineering tasks? *arXiv preprint arXiv:2606.05080*, 2026.

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.

Haotian Ye et al. Structured scaling of AI discovery across diverse scientific domains. *arXiv preprint arXiv:2604.19341*, 2026.

Z.ai. GLM-5.2 model card. https://huggingface.co/zai-org/GLM-5.2, 2026.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, Bang Liu, Chenglin Wu, and Yuyu Luo. Harnessing agentic evolution. *arXiv preprint arXiv:2605.13821*, 2026.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/214308a2d5e3f83ef9ad2739e1cbc46d-Abstract-Conference.html.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Hong Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. SWIFT: A scalable lightweight infrastructure for fine-tuning. *arXiv preprint arXiv:2408.05517*, 2024. URL https://arxiv.org/abs/2408.05517.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, et al. Edgebench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026a.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Cheng Wang, Fengyang Li, Wenkai Jin, Wanxu Liu, Zehao Bing, Bingyang Zheng, Jingyi Chai, et al. EvoMaster: A foundational evolving agent framework for agentic science at scale. *arXiv preprint arXiv:2604.17406*, 2026b.

A LIMITATIONS

Deployment cost. We quantify API usage to characterize the cost of accessing strong models, but do not measure providers' underlying deployment costs, which require finer-grained accounting of hardware utilization, batching, and serving infrastructure. Local small-model serving also introduces costs not fully captured by API token counts. The reduction in strong-model API usage demonstrates the framework's potential to lower API expenditure, although a complete end-to-end cost comparison remains future work.

B ROLE INTERFACES AND PROMPT TEMPLATES

The following templates describe deployment responsibilities; historical training permissions differ as documented in Appendix C.2. They specify the interfaces introduced in Section 3, rather than additional experimental conditions.

B.1 DEPLOYMENT PROMPT TEMPLATES

Figures 5–7 are readable, adapted schematics of the role instructions and invocation fields, not verbatim serialized training examples. Blue fields mark task-dependent content; runtime preambles, full tool schemas and long histories are omitted.

Orchestrator prompt template

System role. You are the central coordinator in a three-role Producer–Verifier system. Plan and decompose the research task, dispatch a Worker to produce changes, dispatch a Verifier to challenge significant outputs, and synthesize the evidence. You control the next action; do not delegate that decision to the Verifier.
Inputs. Task: `{task instruction}` Workspace: `{workspace}` Budget: `{remaining time and turns}` History: `{prior actions, results, and feedback}`
Required procedure. Analyze the objective and success criteria. For every Worker dispatch, include the objective, relevant state, constraints, success criteria, and requested return format. After a significant implementation or result, request adversarial verification. Use new evidence to choose whether to revise, test, switch approach, or submit. Run autonomously within the declared task and budget.
Output contract. Issue the next tool call or provide the final research decision with its supporting evidence and unresolved risks.

Figure 5: **Orchestrator prompt template.** The Orchestrator receives the global task state, delegates bounded work and adversarial review, and retains authority over the next action. Blue fields are populated from the current trajectory.

Worker prompt template

System role. You are a Worker agent. You receive a specific coding task from the Orchestrator and must implement it precisely. Read the relevant code first; make only the requested changes; verify them when permitted; and report what changed and any issue found. Do not make changes beyond the request.
User payload. Objective: `{bounded implementation request}` Background: `{task and prior-result context}` Target: `{files or editable region}` Constraints: `{allowed tools and operations}` Verifier feedback: `{issues to address, if any}` Success criteria: `{observable completion checks}`
Output contract. Return the implemented change, checks performed and their outcomes, files touched, and any unresolved blocker. Do not choose the global research direction or submit the benchmark result.

Figure 6: **Worker prompt template.** Each Worker invocation contains a bounded implementation request, its local context and constraints, and observable completion criteria.

Verifier prompt template

System role. You are a Verifier—an adversarial quality gate in a Producer–Verifier system. Your job is to find problems. Default to skepticism; inspect available evidence; challenge assumptions; and cite concrete files, lines, metrics, or values. Every identified issue must include a proposed discriminating check or repair. A PASS means that you genuinely tried to break the current result and could not.
User payload. Task: `{research objective and metric}` Artifact: `{current implementation or workspace}` Change summary: `{latest Worker report}` Measurements: `{tests, logs, scores, and provenance}` History: `{relevant prior attempts and feedback}` Review focus: `{correctness, effectiveness, and risks}`
Output contract. Separate observed evidence from hypotheses; identify the highest-risk issue; state what remains uncertain; and propose the next check that would distinguish plausible explanations. End with PASS, NEEDS_REVISION, or FAIL. The Orchestrator, not the Verifier, decides and executes the next research action.

Figure 7: **Verifier prompt template.** The Verifier receives checkpoint evidence and returns an adversarial diagnosis, unresolved uncertainty, and a discriminating next check without taking action authority.

B.2 Control Flow and Review Responsibilities

Control flow. Let h_t be the global interaction history. One dispatch-and-review update has the conditional structure

$$\begin{aligned} d_t &\sim \pi_{\mathrm{O}}(\cdot \mid h_t), & w_t &\sim \pi_{\mathrm{W}}(\cdot \mid d_t), \\ q_t &\sim \pi_{\mathrm{O}}(\cdot \mid h_t, w_t), & f_t &\sim \pi_{\mathrm{V}}(\cdot \mid s_t^{\mathrm{V}}, q_t), \\ a_t &\sim \pi_{\mathrm{O}}(\cdot \mid h_t, w_t, f_t). \end{aligned} \tag{5}$$

Here d_t includes the bounded request and Worker-visible context, q_t is a review request, and w_t and f_t are their returns. The Orchestrator chooses both requests and the next action a_t. This is not a mandatory sequence: an omitted invocation contributes an empty return. Checkpoints can follow a Worker return, test, regression, or stalled branch. Figure 8 summarizes the review contract.

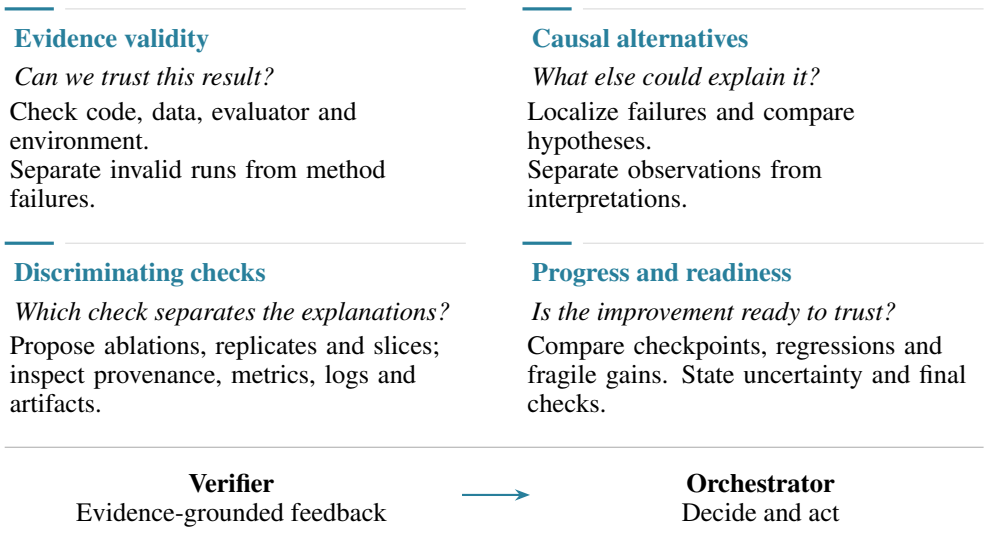


Figure 8: **Verifier interface: four review responsibilities.** This expands the prompt in Figure 7, not an empirical result. The Verifier returns evidence, diagnoses and proposed checks; the Orchestrator retains action authority.

C Data Construction and Training

We document task construction, lifecycle reconstruction, and training settings. Structural checks establish mechanical eligibility, not scientific correctness or independent evidence.

C.1 Task Construction

Research-domain coverage. Figure 9 summarizes the research areas covered by the constructed tasks. These include perception and language, reinforcement learning and robotics, ML systems, scientific and numerical computing, and discrete optimization. The categories describe complementary, potentially overlapping research areas rather than a partition by task count.

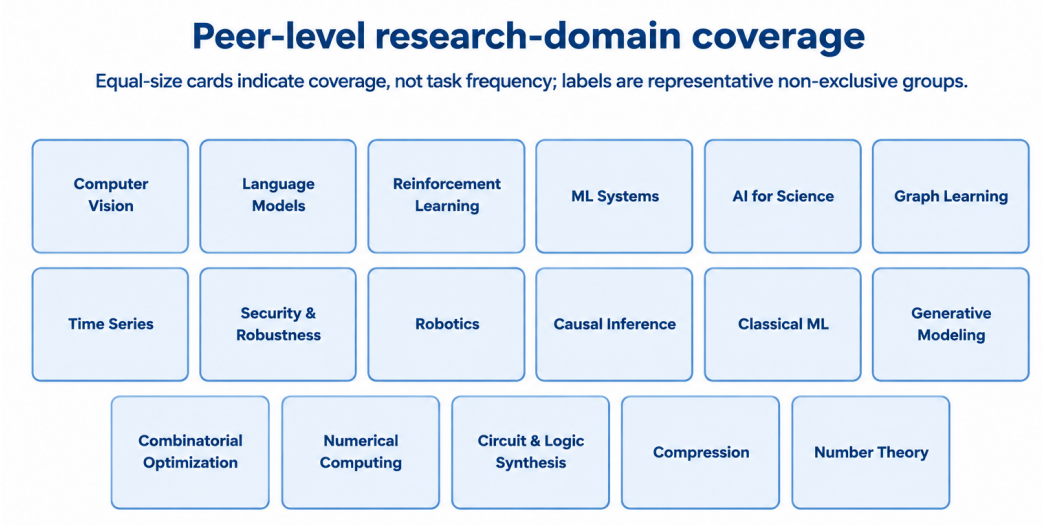


Figure 9: **Research-domain coverage of the constructed tasks.** The 17 representative groups summarize the breadth of task construction. Equal-size cards indicate coverage, not task frequency; a task can involve more than one area.

Source discovery. Public AR task inventories guide a domain map of research problems, dataset families, metrics, methods, and implementation packages. This map exposes coverage gaps and groups parameter-only variants. Each retained source links a paper or repository to an editable implementation, executable measurement, applicable settings, and traceable baselines. Prose without an executable anchor is insufficient.

Normalized record. Each task records its semantic identity, upstream revision, editable component and permitted operations, baselines, public and hidden settings, metric direction and reward transformation, and resource limits. Method sections supply candidate interventions; experiments and released configurations supply evaluation settings. Untraceable quantitative claims are not used as scoring constants.

Workspace and evaluation. The agent-visible workspace contains instructions, editable source code (15–200 lines), and pre-materialized inputs. Evaluation commands and the reference solution remain outside the writable workspace, avoiding evaluator edits or reference artifacts carried into the next rollout. The evaluator scores the completed artifact, not intermediate responses. Table 4 records the acceptance checks and their limits.

Table 4: **Data-pipeline checks for task construction and trajectory recovery.** The two blocks expand the task and trajectory stages in Section 4. Each stage pairs an observed failure with its acceptance rule. The rules establish mechanical validity, not scientific correctness; name and byte-level screening do not exclude semantic overlap.

Stage	Observed failure	Acceptance rule
Executable task construction		
Task identity	Renamed or parameter-only duplicates.	Identify tasks by problem, editable mechanism, data and evaluator.
Source acquisition	Missing metadata, inputs or upstream fidelity.	Preserve provenance; use released code and staged inputs; resolve dependencies.
Specification	Missing code details; mixed metrics or budgets.	Tie each method to code, settings, metric direction and verified reward anchors.
Workspace	Schema errors, latent bugs or evaluator leakage.	Validate code/configs; isolate and hash protected files; normalize paths and clean residual artifacts.
Overlap screening	Benchmark-name or byte-identical overlap.	Name/MD5 screening: 2,359 candidates $\rightarrow$ 2,212 retained; semantic overlap remains possible.
Executable QA	Invalid starts, weak references or timing bias.	Test baseline and hidden reference; restore baseline; serialize and alternate timing rounds.
Lifecycle data construction		
Rollout collection	Missing scores or stale run artifacts.	Require fresh, run-specific outputs; distinguish infrastructure from research failure.
Recovery	Broken agent/tool boundaries or partial endings.	Restore caller/callee and tool pairs; retain compaction provenance; require a final assistant response.
Serialization	Invalid text/JSON or excessive length.	Normalize recoverable records; preserve native tools; reject ambiguity; apply target-template length checks.

Workspace cleaning also removes unsafe links, caches, build outputs, repository metadata and residual solutions.

Terminal scoring. ML tasks use a scoring DSL that maps domain-specific metrics to normalized reward across two or three test settings, some hidden during development. Quality-based optimiza-

tion tasks use reward $= (\text{baseline} - \text{current})/(\text{baseline} - \text{reference})$. Speed tasks use five alternating timing rounds to limit order effects. These scores evaluate completed artifacts and remain distinct from frontier-model Verifier targets.

C.2 Reconstructing Role-Specific Training Examples

Lifecycle reconstruction. We first group events by source run and role invocation rather than treating every log file or assistant message as an independent example. A Worker dispatch becomes one tool call and its completed sub-agent trace becomes the corresponding tool response in the Orchestrator conversation. Verifier invocations are retained as their own conversations. Context-compaction records remain explicit provenance events: the post-compaction state is not silently presented as if it contained the discarded prefix. An assistant tool call without its response, a dangling tool response, consecutive assistant turns without an intervening environment event, or a missing final assistant decision is marked incomplete rather than accepted as a normal trajectory.

Text and tool normalization. Raw logs contain invalid bytes, ANSI escapes, runtime boilerplate, foreign control tokens and malformed JSON. Recoverable text is normalized; the final native format stores ordered `tool_call` and `tool_response` messages, not the original call-ID schema. Missing returns can receive the explicit marker "Tool response unavailable in the source trajectory," not invented substantive output. An audit of an earlier Verifier corpus found examples containing this marker, including tool returns consisting only of the marker. One tool schema is explicitly synthesized because the original was unavailable. These repairs close serialized boundaries, not evidence gaps; balanced counts do not certify correct content matching or fully observed history.

Final training data. We reconstruct role-specific conversations from source sessions, normalize tool interactions, and apply role-specific behavior filters. The final training datasets contain 1,900 Orchestrator, 4,700 Worker, and 7,300 Verifier lifecycles, totaling 13,900 examples for Joint-SFT. These are training examples, not distinct source tasks; all three datasets originate from the same 2,212-task collection. Source-run and task identifiers are retained so audits and future splits can group correlated invocations at the trajectory level.

Historical permission differences. Both system-prompt variants in the audited historical Verifier corpus permit reading, analyzing and fixing issues, explicitly naming Edit. The later deployment prompt instead forbids editing and running tests. The native training traces contain no named Edit or Write call, but retain broad Bash/REPL activity and one BenchSubmit call; this is not evidence that every source invocation was read-only. Call/response counts are balanced, which certifies neither permission compliance nor diagnostic correctness. Thus reinsertion preserves the role position, not literal identity of historical prompts or tool schemas.

C.3 Optimization Settings and Data-Scale Runs

Qwen3.5-9B and Gemma-4-12B use the same training hyperparameters, including a 524,288-token (512K) sequence limit; role data volumes differ. The attention implementation differs: Qwen uses FlashAttention-2 and Gemma uses FlexAttention.

All role checkpoints optimize assistant-token likelihood over their reconstructed lifecycles:

$$\theta_r^* = \arg\min_{\theta} - \sum_{z_i^r \in \mathcal{D}_r^{(512\text{K})}} \sum_{j \in \mathcal{A}_i} \log p_\theta(x_{ij} \mid x_{i,<j}), \tag{6}$$

where $\mathcal{A}_i$ selects supervised assistant tokens and r denotes Orchestrator, Worker or Verifier. Joint-SFT trains their union; terminal task scores evaluate the reassembled system and are not response-level reward labels. Dedicated versus joint SFT changes the role mixture, not lifecycle versus full-trajectory representation; the rollout-source comparison tests a separate aspect of data construction.

Training uses ms-swift (Zhao et al., 2024). The reported Qwen3.5-9B role and joint runs use two epochs, global batch size 32, learning rate 10^{-5} with cosine decay and 5% warmup, a 524,288-token limit, bfloat16, FlashAttention-2 (Dao, 2023), sequence-parallel size 1, gradient checkpointing, and DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) over 32 GPUs. The per-device microbatch

size and gradient accumulation are both 1. We use fused AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 0.1, gradient clipping at norm 1.0, and seed 42. Padding-free execution is enabled. Packing is disabled because the Qwen3.5 linear-attention path used in these runs does not preserve the required variable-length boundaries. We train all language-model parameters, freeze the vision and alignment modules, and save at epoch boundaries. The logged loss-scaling option is `ignore_empty_think`; overlength records use the `delete` truncation strategy rather than prefix truncation.

Data-scale results. Figure 4 evaluates each trained checkpoint on the full MLS-Lite task set. Verifier scores are 28.4, 30.9, 33.4, and 37.1 at 10%, 25%, 50%, and 100% of its training data. Orchestrator scores are 16.5, 18.2, and 19.7 at 25%, 50%, and 100%; Worker scores are 30.7, 31.5, 32.2, and 32.9 at 10%, 25%, 50%, and 100%. Fractions describe training data, not benchmark coverage; differing role corpora do not constitute equal-token controls.

Gemma serialization. The Gemma-4 training implementation uses its native turn, thought-channel, tool-call, and tool-response format rather than Qwen's thinking tags. Adjacent assistant fragments retain their order, and tool names are recovered only when they match the preceding calls. When a tool response is followed by a user message, a loss-masked assistant continuation closes the native turn boundary without discarding either message. Tool observations remain context rather than supervised targets. Gemma uses FlexAttention with native attention masks rather than Qwen's FlashAttention-2 backend, while retaining the same training hyperparameters.

D Evaluation and Reporting Protocols

We specify benchmark inventories, evaluation protocols, and runtime accounting.

D.1 Evaluation Protocols and Result Provenance

Benchmark results. All benchmark endpoint results aggregate the full task set of each benchmark. Our benchmark evaluations report averages over three runs. For external baselines, we report official results directly when available; otherwise, we run the baseline under the same settings and report the three-run average. Gap-recovery and training-return ratios use these endpoints. Research-progress metrics, diagnostic probes, and token statistics complement benchmark endpoint results.

Whole-system result provenance. Table 3 compares training data collected from 25% of the source tasks (12.1, 13.6, 15.6). Training on the full training corpus yields 17.4 with role-structured inference. An older, checkpoint-unmatched Claude Code Joint-SFT run scores 12.2 and is not part of that matched comparison. The retained 9.1 untrained reference supplies context, not a new matched no-training ablation.

How to interpret the comparisons. Benchmark comparisons describe configuration-level outcomes, not the causal value of individual feedback. The primary cross-role training comparison fixes the collection task pool; example counts and training-token exposure follow each role's extracted trajectories. The comparison measures gains from the collected research experience, not equal-compute learning efficiency. Frozen-state scores measure response behavior, filters measure data eligibility, and tokens measure traffic rather than monetary cost.

D.2 Benchmark Scope and Task Distributions

The inventories below define the task populations for benchmark endpoint evaluation.

MLS-Bench-Lite: learning algorithms and model components. MLS-Bench-Lite (Lyu et al., 2026) comprises 30 tasks across 12 research areas. The inventory contains robotics (5); language modeling, vision/generation, ML systems, scientific applications, and time-series tasks (3 each); optimization, classical learning, deep-learning components, and structural/causal tasks (2 each); and reinforcement learning and trustworthy learning (1 each). An agent edits designated parts of an executable ML pipeline, runs experiments, and submits an algorithm or component evaluated under

task-specific settings. The target is method quality across the prescribed evaluation settings; this is our primary controlled role-training benchmark.

AutoLab: research and engineering optimization. AutoLab (Xu et al., 2026) contains 36 tasks: systems optimization (15), puzzles and algorithmic challenges (10), model development (7), and CUDA kernel optimization (4). Agents improve throughput, runtime, resource use, or model quality in a budgeted workspace while satisfying correctness constraints. The official categories describe task organization rather than disjoint learning mechanisms: for example, model development also includes LLM serving optimization. AutoLab extends the role comparisons to project-level research and engineering.

PostTrainBench: developing another language model. The evaluated PostTrainBench version (Rank et al., 2026) pairs four base models—Qwen3-1.7B, Qwen3-4B, SmolLM3-3B, and Gemma-3-4B—with seven target evaluations: AIME 2025, Arena Hard Writing, BFCL, GPQA, GSM8K, HealthBench Easy, and HumanEval. These form 28 model–objective configurations, not 28 unrelated scientific domains. The agent receives an evaluation interface and a budget of ten hours on one H100 to develop a post-trained model. The reported outcome measures that target model, covering mathematics, writing, tool use, scientific knowledge, health, and code generation. Our results here evaluate the execution setting, not a complete three-role specialization study.

EdgeBench: broader feedback-driven problem solving. We use the 51-task public inventory of EdgeBench (Zhu et al., 2026a), not its full 134-task suite. The local task metadata groups these into combinatorial optimization (15), systems and software engineering (12), formal mathematics and theorem proving (8), interactive games and simulators (8), scientific problems and ML (4), and professional knowledge work (4). Outputs range from optimized software and formal proofs to policies, professional deliverables, and progress within an interactive environment. Consequently, not every task is artifact-centered research: this benchmark tests the execution setting on a broader range of environment-feedback loops. Task-specific scores are normalized using the benchmark's scoring convention; our runs use a two-hour interaction budget and should not be equated with longer-budget official results.

D.3 Runtime and Reporting Protocols

Within role-substitution and training ladders, task instructions, benchmark hooks, outcome tools, and run limits are fixed. Benchmark endpoint evaluations follow official time budgets and resource configurations; EdgeBench results use two hours per task. Scores from our own evaluations are averaged over three runs; external baselines follow the reporting rule above. Research-progress analysis uses the configured research budgets, as specified below. Historical MLS-Bench-Lite role-structured harness runs set the interactive limit to $T_i = A_i + D_i$, the sum of the task file's `[agent]` and `[verifier]` timeouts. Here D_i is the benchmark-grading timeout, not a Verifier service allowance. This historical allocation folds grading reserve into the interactive cap, rather than using the agent-only timeout A_i; final evaluation has an additional external buffer. The role-structured harness allows up to 10,000 interaction turns, making wall time the effective stopping constraint. Locally served Qwen3.5-9B and Gemma-4-12B models use a maximum context length of 524,288 tokens (512K). Closed-source models use their official provider configurations; we do not introduce separate context settings for them. All models share a maximum output allowance of 65,536 tokens (64K) per call.

Context management. We automatically summarize conversation history to continue long research sessions within the configured context window. Before a model call, the runtime estimates the conversation length and triggers compaction when it exceeds $C - M - 13{,}000$ tokens, where C is the configured runtime context window and $M = 65{,}536$ is the maximum output allowance per call. Equivalently, the trigger fraction is $1 - (M + 13{,}000)/C$, reserving space for the next response rather than imposing one fixed percentage across models. Compaction replaces earlier history with a summary of decisions, file changes, commands, and current task status, together with up to ten recent messages. The retained suffix is adjusted to preserve message-role boundaries and remove unpaired tool calls and returns.

Search-trajectory accounting. Figure 10 uses valid leaderboard measurements returned within T_i, excluding late or invalid records. The best-so-far curve is zero before the first valid measurement and holds the best observed score between measurements. AUC integrates that curve over elapsed time divided by T_i. New-best count totals strict improvements after the first valid measurement; gain/event averages each improving task's total post-initial gain divided by its improvement count. These are descriptive prefix measurements of one archived rollout per configuration, not outcomes from fresh agent-only-budget runs.

E Research Progress and Inference Usage

We report progress within a research budget, inference token usage, and serving latency as distinct measurements.

E.1 Paired Verifier Progress and Uncertainty

This analysis supports Section 6.1: does Verifier-SFT produce larger improvements than Joint-SFT at the Verifier position? It uses all 30 MLS-Lite tasks, with 19 tasks improving in both conditions for the paired gain/event statistic. Figure 10 shows the paired differences and uncertainty. Orchestrator and Worker remain GLM-5.2. Definitions follow Appendix D; limits are recorded agent-plus-grading allocations, not agent-only budgets. Scores at budget fractions are the best observed scores up to those times, not official final-submission scores.

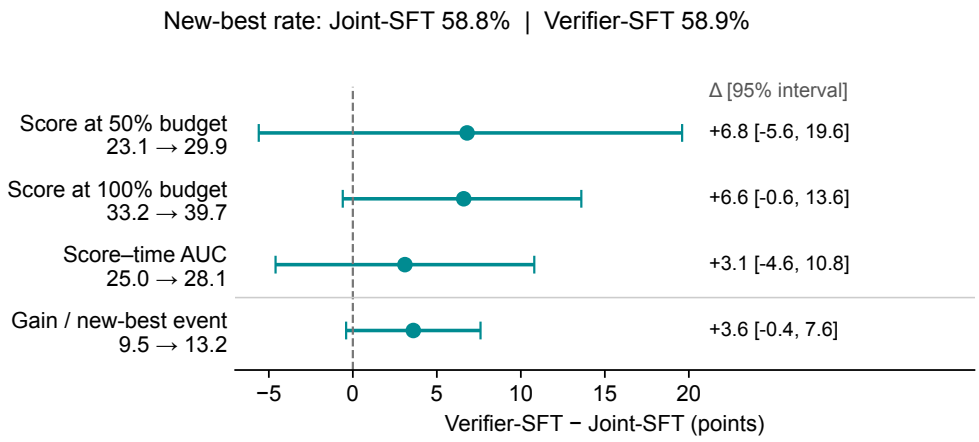


Figure 10: **Verifier-SFT versus Joint-SFT: progress within MLS-Lite runs.** Only the Qwen Verifier checkpoint changes; Orchestrator and Worker remain GLM-5.2. Positive differences favor Verifier-SFT, but all 95% bootstrap intervals cross zero. Labels give Joint-SFT → Verifier-SFT means. Rates and the first three rows use all 30 MLS-Lite tasks; gain/event uses the 19 where both improve. Means and paired differences are rounded independently to one decimal place; subtracting the displayed means can therefore differ from the reported difference by 0.1 point.

Including an initial positive measurement as a new best, mean trajectory-level new-best rates are nearly unchanged (58.9% for Verifier-SFT versus 58.8% for Joint-SFT). Together, these estimates suggest larger successful improvements, not more frequent improvements or reliably earlier convergence.

E.2 Inference Token Accounting

This analysis supports the inference-allocation claim in Section 6.2. Figure 11 summarizes strong-model usage and benchmark performance. Table 5 separates strong- and small-model traffic across role-replacement strategies. Token usage is distinct from complete monetary cost.

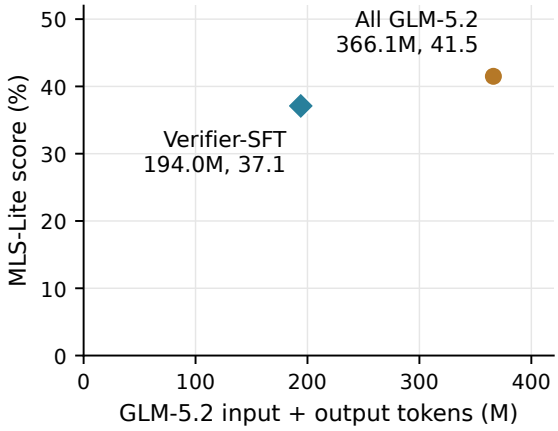


Figure 11: **All-GLM versus Qwen Verifier-SFT: token use and MLS-Lite score.** The horizontal axis counts GLM-5.2 tokens; the vertical axis shows task performance. Both axes summarize MLS-Lite evaluation.

Model-family accounting. Table 5 reports inference usage by model family on MLS-Lite.

Table 5: **Model ledger: strong- versus small-model inference usage.** O/W/V identifies the replaced Orchestrator/Worker/Verifier; the other roles remain GLM-5.2. Column groups identify the replacement model: Qwen3.5-9B or Gemma-4-12B. Base is untrained and SFT is role-trained. I/O denotes input/output tokens in millions. The Qwen Verifier-SFT totals match Figure 11; these are token counts, not monetary costs.

	Qwen3.5-9B		**Gemma-4-12B**	
Strategy	**GLM I/O**	**Small I/O**	**GLM I/O**	**Small I/O**
All-strong	359.0 / 7.1	0.0 / 0.0	359.0 / 7.1	0.0 / 0.0
O: Base	140.0 / 2.5	260.3 / 1.2	153.1 / 2.6	230.7 / 1.2
O: SFT	180.1 / 4.2	201.2 / 4.1	170.6 / 4.2	181.8 / 4.1
W: Base	123.6 / 2.7	53.3 / 0.6	144.2 / 2.2	43.1 / 0.7
W: SFT	103.6 / 2.3	1.6 / < 0.1	122.9 / 2.3	24.1 / 0.2
V: Base	200.3 / 4.1	30.4 / 1.3	179.5 / 3.4	13.9 / 0.2
V: SFT	190.3 / 3.7	16.1 / 0.7	161.6 / 2.8	34.3 / 1.1

Accounting scope. Figures 1 and 11 show the Qwen Verifier-SFT deployment, whose GLM input/output usage is 190.3M / 3.7M in Table 5. Their 194.0M total is 47.0% below the all-strong-model system's 366.1M. Gemma Verifier-SFT uses 161.6M / 2.8M GLM input/output tokens, totaling 164.4M, or 55.1% fewer than the same all-strong reference. The equal-weight mean reduction across Qwen and Gemma is 51.1%, calculated before rounding the individual percentages. Its small-model usage totals 35.4M tokens (34.3M input and 1.1M output), compared with Qwen's 16.8M (16.1M input and 0.7M output). Small-model token usage describes the recorded inference calls. API-reported token counts are not a common monetary unit across API and local serving; full cost also requires missing usage and local GPU-seconds.

E.3 Serving Latency and Delivery

Higher raw token throughput does not imply faster usable feedback in this probe. Across three repetitions of 30 matched prompts at concurrency 15 and a 64K output limit, GLM-5.2 returns 90/90 usable responses versus 27/90 for Verifier-SFT. Median block-level successful-call P50 is 70.5 versus 187.7 seconds. The local model's higher raw throughput (995.7 versus 664.7 tokens/s)

includes long, length-terminated outputs. Thus it does not demonstrate faster useful feedback. This exploratory serving comparison is distinct from the matched research-progress analysis.

F Diagnostic Probes and Their Limits

These probes examine how supplied hypotheses and evidence affect review, separately from benchmark endpoints. Unless otherwise noted, judgments are model-assisted and condition-blind; response-quality scores are not downstream rewards. Paired uncertainty uses 20,000 checkpoint-level bootstrap resamples with seed 42, without multiplicity correction.

F.1 How Does the Quality of an Upstream Hypothesis Affect Review?

Evidence-supported hypotheses yield fewer unsupported claims than contradicted hypotheses. Neither differs conclusively from supplying no hypothesis. Table 6 changes the supplied hypothesis while holding the Verifier model and evidence fixed.

Matched conditions. At each of 30 frozen MLS-Bench-Lite checkpoints, model-assisted annotation identifies a historical hypothesis supported by the evidence and one contradicted by it. GLM-5.2 reviews identical isolated evidence with no hypothesis, the supported hypothesis, or the contradicted hypothesis. Hypotheses are labeled unverified claims, not instructions; all 90 generations share the decoding configuration. This tests sensitivity to claim quality, not the prevalence of harmful natural messages.

Table 6: **Hypothesis probe: supported, contradicted, or no upstream claim.** The GLM-5.2 Verifier reviews the same frozen states under three supplied-hypothesis conditions. Calibration scores whether claims track the evidence (0–4); the other columns count unsupported claims and critical omissions per response.

Supplied hypothesis	Calibration (0–4) ↑	Unsupported claims ↓	Critical omissions ↓
None	1.8	1.9	1.8
Evidence-supported	2.2	1.6	1.6
Evidence-contradicted	1.8	2.0	1.8

Supported hypotheses yield 0.47 fewer unsupported attributions per response than contradicted hypotheses (95% paired bootstrap CI $[-0.83, -0.13]$), with 0.23 fewer critical omissions ($[-0.50, -0.07]$). Exact sign tests excluding ties give $p = .057$ and $p = .062$, respectively. Neither supplied condition differs conclusively from no hypothesis. Upstream interpretations should therefore be checked against evidence rather than removed by default or treated as facts.

Judging and uncertainty. DeepSeek-V4-Pro serves as the judge and receives each matched triplet in a deterministic shuffled order with condition names hidden, together with the complete trajectory and frozen evidence used for source checking. It codes 0–4 actionability and calibration, lists supported findings, unsupported causal attributions, and decision-critical omissions, and records whether recommendations diverge. Reported error columns are list lengths per response. Differences use the shared paired-bootstrap protocol; two-sided sign tests exclude ties. A separate condition-blind Kimi-K3 pointwise pass covers all 90 responses but is not used for the primary paired error-count claim. The append-only audit resolves retries by latest record ID and verifies 90/90 effective generations, 90/90 Kimi judgments, 30/30 DeepSeek triplet codings, exact condition mappings, and zero malformed or missing-key rows. The annotations and final coding are model-assisted rather than human labels, and the experiment evaluates diagnosis quality rather than downstream task reward.

G TRAJECTORY CASES AND FAILURE ANALYSIS

Case selection. We inspect 24 complete trajectories spanning both controlled benchmarks, positive and negative score gaps, and successful and failed Verifier delivery. These selected cases illustrate mechanisms, not their prevalence or causal effects.

G.1 WHERE ARE VERIFIER-TRAINING GAINS CONCENTRATED?

We compare AutoLab runs using the base Qwen3.5-9B Verifier and its Verifier-SFT checkpoint, grouping tasks by the benchmark's four task families. Table 7 summarizes score differences rather than repeating endpoint scores.

Larger gains on CUDA kernels, smaller gains on systems. CUDA kernels have the largest mean gain (+23.1), followed by model development (+14.6) and puzzles (+7.1). Systems optimization has the smallest mean gain (+3.2). All four domain averages improve, but the size of the benefit differs substantially across research settings.

Table 7: **Domain-level changes after Verifier training on AutoLab.** Changes are SFT minus Base, in score points; task families follow the benchmark taxonomy. These runs provide 35 paired scores; `llm_online_serving` lacks a Base score and is not imputed. This run-level breakdown complements the multi-run endpoint results in Table 2.

Task family	Tasks	Mean change
CUDA kernels	4	+23.1
Model development	6	+14.6
Puzzles	10	+7.1
Systems optimization	15	+3.2

The domain averages describe configuration-level outcomes, not uniform gains on every task. The model-development comparison includes a baseline execution failure; an unchanged puzzle result also includes a strong-model feedback call in the Base run. These averages therefore characterize the observed distribution of gains without isolating diagnostic quality as their cause.

MLS-Lite: gains differ across research domains. We additionally compare archived Base and Verifier-SFT Qwen3.5-9B deployments with Opus 4.6 in the surrounding roles. The runs match in execution mode, round limit, stopping strategy, and per-task time limit; the Base and SFT runs use AWS and Vertex model endpoints, respectively. Table 8 reports all domain groups from these runs, rather than selecting only positive changes. Among domains represented by multiple paired tasks, optimization (+30.7), time series (+15.3), and AI for science (+11.5) show larger gains than language-model tasks (+0.8). The negative domain averages are all based on single tasks and do not establish general domain weaknesses. Across the paired tasks, the mean changes from 27.7 to 36.3; this historical comparison is distinct from the GLM-5.2 deployment results in Table 2.

Feedback utility may depend on the research problem. One possible explanation is that review is most useful when experimental evidence points to actionable revisions, such as changes to an optimization procedure or model implementation. In contrast, quantitative finance and world-model planning may depend more strongly on domain priors and the initial choice of modeling assumptions or planning strategy. If these upstream choices are the bottleneck, improving the Verifier's diagnoses alone may offer limited benefit without corresponding improvements in decision-making and execution. The observed domain differences motivate this hypothesis, rather than establish a causal mechanism or a ceiling on Verifier training.

The ML-systems Base trace also contains strong-model feedback calls, so its near-zero change is not a pure small-Verifier contrast.

Table 8: **Domain-level changes in archived MLS-Lite Verifier runs.** Changes are SFT minus Base in score points, grouped by task-domain prefixes. The task counts refer to paired scores; 27 of 30 tasks have valid scores in both runs. Missing or invalid scores are not imputed. Single-task groups should be read as observations rather than broad domain estimates.

Domain	Tasks	Mean change
Reinforcement learning	1	+58.5
Optimization	2	+30.7
Deep learning	1	+25.6
Time series	2	+15.3
AI for science	3	+11.5
Graph learning	1	+10.8
Classical ML	2	+8.7
Computer vision	4	+8.6
Robotics	3	+6.7
Language models	3	+0.8
ML systems	1	0.0
Security	1	−4.2
Causal inference	1	−6.6
World-model planning	1	−11.2
Quantitative finance	1	−41.0

G.2 What Changes along Role-Replacement Trajectories?

The traces distinguish configured roles from actual calls, delivered feedback and direct Orchestrator implementation.

Verifier-SFT versus Joint-SFT: progress within a run. The inspected trajectories illustrate the distinction between improvement frequency and improvement size. We use the aggregate analysis in Section 6.1 to quantify that distinction, rather than interpreting individual score jumps as evidence of a general trend.

Worker Base-to-SFT: dispatch changes complicate attribution. Worker traces show changes in dispatch frequency, runs that do not invoke the configured Worker, and direct implementation by the Orchestrator. These observations motivate dispatch-matched comparisons rather than attributing configuration-level score differences solely to Worker weights.

Verifier substitutions: competing explanations. The inspected runs also differ in execution failures, iteration counts, implementation choices, and successful Verifier use. These observations help interpret configuration-level outcomes but do not establish the cause of domain-level differences.

G.3 A Logged Feedback-Adoption Case

This historical Qwen3.5-9B Verifier-SFT run with an Opus 4.6 backbone on `cv-vae-loss` illustrates feedback entering the research loop, rather than a comparison of trained and untrained Verifiers. At 296.8 minutes, the Verifier returns its review; the Orchestrator explicitly adopts recommendations and edits the loss at 297.1–297.3 minutes. The changes delay GAN activation from step 5,000 to 10,000, set the LPIPS weight to 1.5, and reduce the frequency-loss weight from 0.1 to 0.05. The subsequent evaluation improves the normalized score by 1.0 point; medium- and large-scale rFID improve, while small-scale rFID worsens. Multiple edits occur together, so their individual effects and the counterfactual outcome without feedback remain unknown.

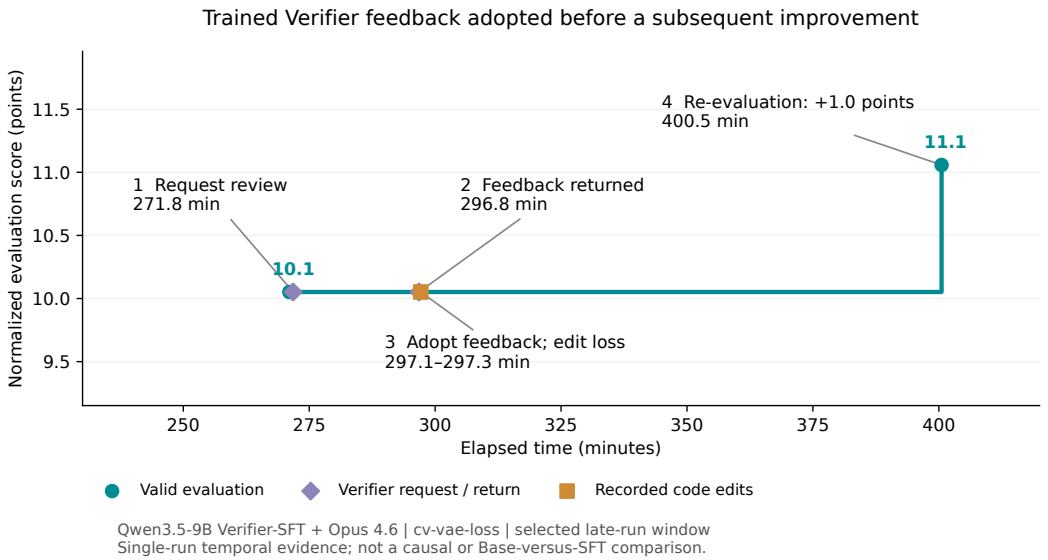


Figure 12: **Feedback-adoption case: Qwen3.5-9B Verifier-SFT on MLS-Lite `cv-vae-loss`.** The horizontal axis is elapsed runtime; the vertical axis is the normalized evaluation score. Circles mark valid evaluations, diamonds mark Verifier dispatch and return, and the square marks recorded edits. Event markers between evaluations sit at the last observed score, not a newly measured score. The plot shows a late-run window and retains the actual timestamps; it does not imply immediate improvement when code changes. This selected single-run case supports feedback adoption, not a causal effect or a training-before/after efficiency comparison.

G.4 Why Can Joint Training Fail in an All-Small-Model System?

Table 9 groups nine score-−1 traces by attribution, recovery, revision coherence and stopping. They use the full-data all-role Joint-SFT checkpoint, not the 25%-source-task checkpoint in Table 3, and do not isolate Orchestrator ability.

Table 9: **Failure cases from all-role Joint-SFT deployment.** All three roles use jointly trained Qwen3.5-9B. Score-−1 traces expose recovery and artifact-validity problems; shared environment failures are not treated as model deficits.

Capability	Failure pattern	Task-level evidence
Failure attribution	Failure layer unresolved.	`cv-vae-loss`: unavailable data. `cv-3dgs-densification`: missing image path. `llm-rl-importance-sampling`: shared benchmark defect, not a model gap.
Evidence-driven recovery	Repeats tests without verified repair.	`quant-concept-drift`: Joint repeats the Qlib missing-day error; GLM repairs it (6.9%). `ai4sci-inverse-diffusion-algo`: recursive symlink remains unresolved.
Revision coherence	New errors or repeated interface violations.	`ts-exogenous-forecast`: syntax, missing-class, string and indexing errors. `robo-diffusion-guidance`: shape and API errors. `llm-ptq-algorithm`: repeated tensor-size violation.
Budget and stopping	No fallback to a valid artifact.	`robo-diffusion-policy`: 21,900-second timeout; GLM repairs a tensor error and scores. `llm-rl-importance-sampling`: both time out; not evidence of a model gap.