AI DESIGNS AND TRAINS TOOL-USING AGENT TEAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing a tool-using agent requires deciding what to test, how to organize work, and how to learn from execution. These decisions interact: a change in system design alters both the failures that remain and the experience available for training. We introduce RESIDUALFORGE, an AI-directed framework that connects executable task construction, failure-guided system design, and native policy training. The designer refines a single actor, tests changes to its skills and division of work, and records which cases each change repairs or regresses. The selected design is then fixed while its policies learn from trajectories that retain actual tool observations and role-to-role handoffs. Across ClawEval, QwenClawBench, and TeamBench, we evaluate complete systems, their development components, and transfer across execution harnesses. On QwenClawBench, on-policy distillation improves the selected Qwen3.5-9B system from 34.36 to 37.42. Removing team handoffs from training context causes the largest component-study drop on ClawEval and QwenClawBench, and pooled training rollouts improve performance in an unseen harness. System selection also retains a single actor on TeamBench when larger candidates perform worse. These results connect autonomous engineering decisions to the quality of the resulting system and the execution experience used to train it.

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.

1 INTRODUCTION

Building a tool-using agent involves repeated decisions about what to test, how to organize execution, and what to train. An engineer runs a model in its intended environment, inspects failures, revises instructions or skills, and considers whether a new role or policy update would help. Each decision changes the system available for the next experiment. Automating this process requires an AI designer to use execution evidence to choose and evaluate those changes.

The workflows in Figure 1 show why these choices matter. Updating code can require regenerating dependent files; restoring a scheduled job requires verifying the resulting state; producing a report requires finding inputs and checking several artifacts. The same failed task can call for different interventions. Missing a file may require better discovery instructions, repeated verification failures may motivate a dedicated role, and unreliable tool use may call for training. The designer must choose an intervention from the observed behavior and test whether it improves the system.

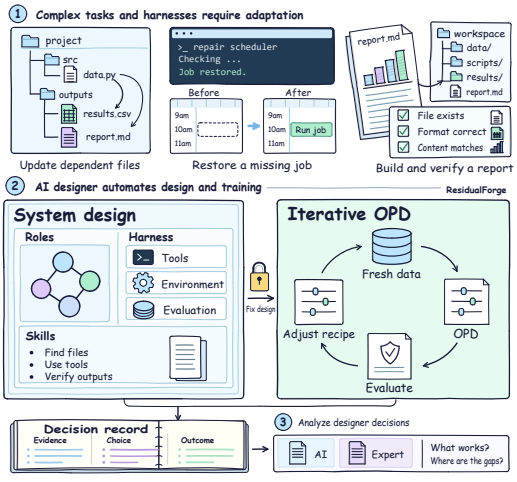


Figure 1: **Autonomous agent engineering.** Complex workflows motivate decisions about skills, roles, and training. An AI designer tests system revisions, fixes the selected design, and improves its policies through native execution. Decision records connect available evidence, proposed changes, and outcomes; shared saved states provide the inputs for comparing AI and expert decisions.

System design also determines what the actors need to learn. In the scheduler example, a role that receives a restored workspace needs to verify the repair. Its training context must include the updated state and the preceding handoff. When an upstream policy changes, downstream roles encounter a different distribution of inputs. Development experience, division of work, and policy learning are thus coupled: useful training depends on the system that will execute it, and useful system revisions depend on failures exposed by executable tasks.

Recent methods automate agent programs and workflows (Hu et al., 2025; Zhang et al., 2025), harness revision (Lee et al., 2026), and multi-agent orchestration (Ke et al., 2026a). Native-harness training connects policy updates to stateful execution (Yu et al., 2026), while autonomous post-training and joint prompt–weight optimization extend the decisions delegated to an AI system (Shi et al., 2026; Soylu et al., 2024; Ziems et al., 2026). We study the complete development process linking these decisions: constructing useful experience, selecting a reusable system, and training its policies in the execution contexts that the selected system produces.

We introduce RESIDUALFORGE, an AI-directed framework for designing and training tool-using agent teams. The user supplies a goal, actor model, execution harness, permitted operations, and resources. The designer first constructs executable development tasks and refines a single actor. It then uses residual failures to propose changes to prompts, skills, responsibilities, communication, and activation. Each tested change contributes a record of the cases it repairs and regresses. After system selection, the design is fixed while the policies learn from native trajectories with the actual tool observations and handoffs. Figure 2 organizes this process into four stages, with on-policy distillation (OPD) illustrating the online training loop.

We evaluate the returned systems on ClawEval, QwenClawBench, and TeamBench. System search improves the Qwen3.5-9B ClawEval endpoint from 25.93 to 32.10 pass^3, and OPD improves Qwen-ClawBench Score from 34.36 to 37.42. Component studies examine how task construction, revision history, and training context affect performance; removing team handoffs causes the largest drop on ClawEval and QwenClawBench among these interventions. Cross-harness experiments test how training experience transfers when the execution interface changes. The selected TeamBench design keeps a single actor after larger candidates underperform, making system size an outcome of development.

The decision records connect these system-level results to the choices made during development (Figure 1, bottom). They retain the evidence available to the designer, the proposed change, and its effects on individual tasks. This makes it possible to examine how the designer responds to remaining failures, preserves successful behavior, and prepares training experience. Shared saved states also define a common input for comparing AI and expert choices.

Our contributions are threefold: (1) an autonomous engineering formulation that links development experience, system design, and policy learning; (2) RESIDUALFORGE, a four-stage framework that carries execution evidence through revision records and role-conditioned trajectories; and (3) an evaluation across three environments, with component and cross-harness studies explaining how these choices affect the resulting systems.

2 RELATED WORK

Automated agent and harness design. Agent-design methods search over different parts of an executable system. ADAS, AFlow, and AgentSquare optimize programs or modular workflows (Hu et al., 2025; Zhang et al., 2025; Shang et al., 2025); DSPy, MIPRO, and GEPA optimize instructions and demonstrations (Khattab et al., 2023; Opsahl-Ong et al., 2024; Agrawal et al., 2026). Meta-Harness uses prior code, scores, and traces to propose harness revisions, Self-Harness connects edits to observed weaknesses and regression tests, and SoL-Pi scales harness exploration (Lee et al., 2026; Zhang et al., 2026a; Liu et al., 2026). BetterTogether and multi-module GRPO connect configuration search to policy updates (Soylu et al., 2024; Ziems et al., 2026). These methods establish the design variables and feedback available to an automated developer. RESIDUALFORGE connects their selection to executable task construction and subsequent training under the selected system. Our component studies evaluate these decisions at the level of the returned system, with model and resource controls motivated by harness evaluations (Fan et al., 2026).

Multi-agent organization and learning. AutoGen and MetaGPT provide collaborative execution patterns (Wu et al., 2023; Hong et al., 2024). MAS-ZERO constructs compositions for individual problems; MAS-Orchestra learns orchestration and studies which task structures benefit from teams (Ke et al., 2025; 2026a). Self-Organizing Agent Teams learn reusable collaboration strategies for a fixed membership (Pappu et al., 2026). SiriuS improves policies through bootstrapped reasoning, and AgentFlow trains a planner within a live multi-module execution loop (Zhao et al., 2026; Li et al., 2026). These methods connect local agent behavior to joint execution. RESIDUALFORGE selects a reusable division of work, including team membership, on development tasks, then trains policies using the histories and handoffs produced by that division. Its revision records track both repaired and regressed cases, including coordination failures emphasized by multi-agent failure analyses and evolving-harness evaluations (Cemri et al., 2026; Ke et al., 2026b).

Native training and autonomous improvement. OpenForgeRL and Agent Lightning connect training to native agent execution through interfaces that support stateful tools and complex workflows (Yu et al., 2026; Luo et al., 2025; He et al., 2026). AgentEvolver, Agent0, and EnvCraft construct experience or executable environments for learning (Zhai et al., 2025; Xia et al., 2025; Zeng et al., 2026). On-policy distillation provides teacher feedback on student-generated sequences (Agarwal et al., 2024; Gu et al., 2024), while A-Evolve-Training delegates decisions across post-training rounds (Shi et al., 2026). SkillRL and OpenClaw-Skill connect reusable skills to policy improvement (Xia et al., 2026; Lin et al., 2026); DGM, Hyperagents, and Theseus examine broader self-improvement processes (Zhang et al., 2026b;c; Duan et al., 2026). In our framework, training follows system selection. Fixing prompts, skills, and routing during training preserves the selected organization, while fresh native rollouts capture the role inputs induced by changing policies.

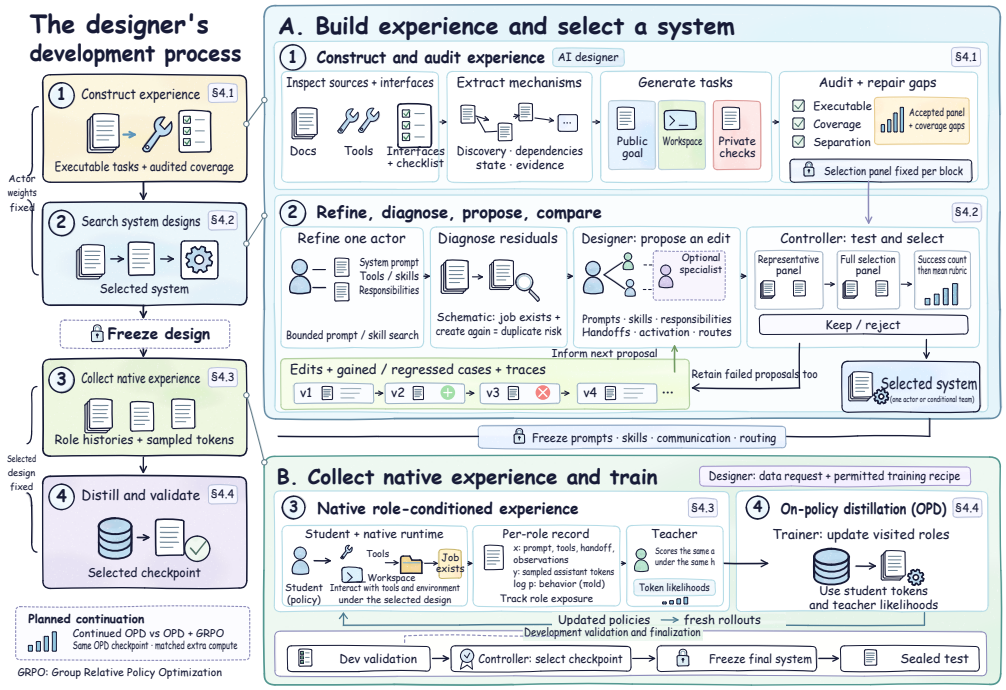


Figure 2: **The four-stage development process. A: build experience and select a system.** Stage 1 constructs and audits executable tasks. Stage 2 tests system revisions and records repaired and regressed cases. **B: collect native experience and train.** With the selected design fixed, Stage 3 records role-visible histories and learning signals; Stage 4 updates eligible policies and validates checkpoints. The illustrated OPD loop collects fresh student trajectories after updates and obtains teacher likelihoods on the same histories. The final system is evaluated on sealed tasks.

3 AUTONOMOUS AGENT ENGINEERING

The development problem is to return an executable agent system for a supplied task environment. The *user* specifies the goal, initial actor models, resources, and permitted operations. The *designer*

constructs development experience and proposes system revisions or training specifications. A configured *controller* executes comparisons, enforces the supplied limits, and applies selection rules. The *actors* execute tasks using the resulting system.

A task consists of a public goal x, an initial workspace w_0, and a private grader g. The native execution harness H provides the tool interface, observations, state transitions, and execution limits. A system $S = (V, P, \rho, M)$ specifies its roles V, prompts and skills P, routing ρ, and communication rules M. Its policy parameters are $\theta = \{\theta_i : i \in V\}$.

The designer acts across development experiments; actors act within an execution. Given a delivered history h_t, routing selects $i_t = \rho(x, h_t)$ and the active policy produces $a_t \sim \pi_{\theta_{i_t}}(\cdot \mid P_{i_t}, h_t, H)$. The environment returns o_{t+1} and updates the workspace, and the grader evaluates the completed trajectory and state. After inspecting development outcomes, the designer can revise the experience, propose a new S, or specify an update to θ.

The output is a selected design S^*, its trained policies θ^*, and the development history connecting them to the tested changes. We measure the quality of this complete system and record its resource use. Appendix G gives the campaign configuration; Appendix E defines the source partitions used for development and evaluation.

4 RESIDUALFORGE

RESIDUALFORGE develops a system in two phases (Figure 2). With actor weights fixed, the designer constructs executable experience and searches system designs. With the selected design S^* fixed, the controller collects native trajectories and updates the configured policies. These phases form four stages: construct experience, search designs, collect role-conditioned trajectories, and train. Revision records connect the first two stages; native execution histories connect the selected design to learning.

4.1 CONSTRUCT EXECUTABLE DEVELOPMENT EXPERIENCE

The designer uses source tasks and their harness interfaces to identify the execution mechanisms that development experience must cover. These include discovering relevant files, updating dependent artifacts, carrying state across actions, and verifying completion. Each generated task contains a public goal, an initial workspace, and private checks. The public goal specifies the desired outcome; the actor determines the tool sequence and solution.

The controller audits executability, environment dependence, duplicate structure, and separation between the public request and private checks. Coverage is tracked by execution mechanism as well as topic. For example, changing a finance task into a coding task preserves the same mechanism if both only require reading, comparing, and writing a file. A gap in debugging or state verification triggers targeted task generation within the repair budget. The stage returns an accepted development panel and its coverage record.

Diagnostic probes can expand as new failures appear, while the selection panel stays fixed within each comparison block. Generated variants retain their source identity, and development, checkpoint validation, independent audit, and sealed testing follow the campaign's source partitions (Appendix E).

4.2 SEARCH DESIGNS USING RESIDUAL FAILURES

Search starts from a single actor or a validated parent system. The designer refines its instructions and skills, examines the failures that remain, and proposes revisions to responsibilities, communication, or routing. A proposed role specifies what it receives, what it does, and when it is activated. The harness supplies the available delegation interface, so the division of work can vary with the environment.

Each proposal is grounded in task-level execution evidence: delivered role inputs and outputs, tool results, paired traces, and the grader feedback permitted during development. The revision record links a proposed change to behaviors it repairs or regresses and to successful interactions worth

preserving. Rejected changes remain available for later diagnosis. Actors receive the public task context; development-only grading evidence stays in the designer's diagnostic record.

The controller first compares revisions on a representative panel of failure cases, then evaluates the selected proposal on the full development panel. For a parent S and candidate S', it records the numbers of gained and regressed cases,

$$G(S', S) = \sum_x \mathbf{1}[b_x(S') = 1, b_x(S) = 0], \qquad R(S', S) = \sum_x \mathbf{1}[b_x(S') = 0, b_x(S) = 1], \quad (1)$$

where b_x is the task's binary success verdict. These counts retain changes that an aggregate score can hide. The campaign specifies pass-first or mean-score-first selection and can additionally require $R = 0$. A validated parent is stored separately from the working candidate. Search returns the best completed design under the promotion rule, retaining the parent when larger candidates fail to improve it. Proposal limits and the stopping rule are fixed by the campaign.

4.3 Collect native, role-conditioned trajectories

Native trajectories are collected by running the selected system through the harness's actual tools and workspace state. The design fixes the instructions, observations, and upstream handoffs that each role receives. The recorder preserves that delivered history for every assistant call, together with its role, policy version, generated tokens, and the behavior-policy likelihoods needed for online updates. In the scheduler example, a verification role therefore learns from the restored workspace and actual repair handoff.

The trajectory source depends on the learning recipe: supervised fine-tuning (SFT) uses teacher executions, while OPD and Group Relative Policy Optimization (GRPO) use current-student executions. All tool observations come from the native environment. Each configured trainable role learns from its own assistant outputs, including serialized tool calls; instructions, tool returns, and other roles' outputs form the conditioning context. Private grading information supplies the permitted outcome signal and is excluded from supervised completions.

Let $V_{\text{train}} \subseteq V$ be the configured trainable roles. The controller records which roles are visited and provide valid training calls. Online updates are followed by fresh rollouts, exposing downstream roles to the handoffs produced by the updated policies. Offline SFT uses its collected teacher corpus. This distinction determines whether training follows the system's changing execution distribution.

4.4 Train policies within the selected design

The learning recipe specifies its trajectory source, supervision, trainable roles, and update budget. Prompts, skills, routing, and communication remain fixed while policies are updated. The controller collects experience, runs optimization, evaluates checkpoints, and redeploys the selected policies for further online rollouts.

Teacher-trajectory SFT. A teacher executes the selected system using its public instructions and native tools. For the complete, execution-valid teacher calls $\mathcal{D}_i^T$ of role i, the student minimizes $\mathcal{L}_{\text{SFT},i} = -\mathbb{E}_{(h,y)\sim\mathcal{D}_i^T} \sum_t \log \pi_{\theta_i}(y_t \mid h, y_{<t})$. The target mask covers that role's assistant tokens. The recipe retains valid complete calls without filtering by task grade; truncated calls and infrastructure-failed executions are excluded.

On-policy distillation. Current student policies generate native trajectories, and a frozen teacher scores their sampled assistant tokens under the same role-visible histories. The sampled reverse Kullback–Leibler (KL) objective is

$$\mathcal{L}_{\text{OPD}} = \mathbb{E}_{i\in V_{\text{train}},\, h\sim d_\theta^{S^*,H},\, a\sim\pi_{\theta_i}} \left[\log \pi_{\theta_i}(a \mid h) - \log \pi_T(a \mid h)\right], \quad (2)$$

where $d_\theta^{S^*,H}$ is the history distribution induced by the selected design, native harness, and current policies, and a is an assistant token. Eligible visited roles are updated before the next rollout batch. Teacher scoring uses the exact student tokens and histories, so its feedback addresses the decisions the current system actually makes (Agarwal et al., 2024; Gu et al., 2024).

Outcome-reward GRPO. The current policy produces a group of native executions for each development task. Grader rewards r_j yield group-relative advantages $A_j = (r_j - \bar{r})/s_r$, with s_r the sample standard deviation. Zero-variance groups are skipped. The update uses the clipped policy surrogate with a fixed-reference KL penalty (Shao et al., 2024),

$$\mathcal{L}_{\text{GRPO}} = \mathbb{E}[-\min(q_t A_j, \text{clip}(q_t, 1-\epsilon, 1+\epsilon)A_j) + \beta k_t], \tag{3}$$

where $q_t = \pi_{\theta_i}(a_t \mid h_t)/\pi_{\text{old},i}(a_t \mid h_t)$ and k_t estimates the current/reference KL divergence. Loss is applied to eligible assistant tokens, averaged within calls, and weighted by inverse call count within each episode. Confirmed infrastructure or grader failures are excluded from outcome rewards. The configured continuation starts from an OPD checkpoint; the campaign specifies which roles are updated. After the selected learning schedule, the controller freezes the final system and evaluates it on sealed tasks. The saved checkpoint retains its role mapping and training lineage. Algorithm 1 summarizes the development process, and the appendix records campaign-specific backends and execution settings.

5 Experiments

We evaluate the systems produced by the complete development process, then examine how the decisions made during development affect their quality. The experiments address three questions: how the returned systems perform, how system design and training work together, and how the learned behavior transfers across execution harnesses.

5.1 Setup

The principal experiments use Qwen3.5-9B with thinking enabled and Qwen3.5-122B as the teacher. We evaluate ClawEval's general-task track (revision `0543559`), QwenClawBench under native OpenClaw execution and workspace grading, and TeamBench. Development tasks and training derivatives inherit their original source's partition; protected evaluation sources are kept separate from development (Appendix E). On TeamBench, the selected system and its single-actor control both use the full-access interface. Table 1 also includes external reference models and historical Qwen3-8B configurations; Appendix H records their sources and settings.

ClawEval measures both reliability and coverage across three trials: pass^3 is the percentage of tasks passed in all three trials, and pass@3 is the percentage passed at least once. QwenClawBench Score averages native fractional task scores, and TeamBench Pass averages strict binary verdicts. All values use a 0–100 scale. Appendix B gives the metric formulas and trial handling. The component and transfer studies use the controls and resource settings specified in Appendix F.

5.2 Performance of the returned systems

Table 1 places the returned systems alongside external reference models and historical local systems. Its three blocks distinguish these groups, and the Qwen3.5-9B block follows development from the initial actor to the selected design and trained policies. OpenForge-Claw uses Qwen3-30B-A3B as its backbone.

Within the Qwen3.5-9B block, design search raises ClawEval pass^3 from 25.93 to 32.10 and pass@3 from 58.02 to 62.96. On TeamBench, the selected system reaches 28.15 Pass, compared with 25.93 for the initial control. The designer retains a single actor after three larger candidates score below its parent on the development panel. System size is thus a decision made during development, and the selected system can remain small when additional roles reduce quality.

5.3 Learning within the selected design

With the selected design fixed, OPD supplies teacher feedback on the student's own executions. OPD raises QwenClawBench Score from 34.36 to 37.42 and TeamBench Pass from 28.15 to 30.37. On ClawEval, the OPD endpoint reaches 35.80 pass^3 and 66.67 pass@3. These outcomes show how the selected design performs after learning from its own native execution histories.

The source of the learning signal affects the outcome. Training on teacher demonstrations (SFT) reaches 34.42 on QwenClawBench, close to the selected system's 34.36, while OPD reaches 37.42.

System	ClawEval pass^3	ClawEval pass@3	QCB Score	TeamBench Pass
External reference entries (sources in Appendix H.2)				
Claude Opus 4.6	70.8	80.8	59.5	28.9
Claude Sonnet 4.6	68.3	81.4	60.5	27.8
GPT-5.4	60.2	75.8	56.7	27.8
Gemini-3.1 Pro	55.9	80.8	51.2	28.9
Gemini-3 Flash	48.4	67.1	49.8	25.6
Qwen3.5-397B-A17B	57.8	70.8	54.2	22.2
Qwen3-32B	6.8	31.7	14.6	1.1
Qwen3-14B	4.4	8.1	12.1	2.2
Qwen3-8B	3.1	6.2	11.5	0.0
Qwen3-30B-A3B-Thinking	14.3	39.8	21.8	11.1
Qwen3-Coder-30B-A3B	30.4	49.7	24.3	15.6
OpenForge-Claw (SFT)	21.7	52.1	32.1	16.7
OpenForge-Claw (SFT+RL)	31.7	55.9	33.7	17.8
Qwen3-8B				
Single-actor control	3.1	18.6	16.4	0.7
RESIDUALFORGE, selected design	5.0	26.7	24.3	4.4
RESIDUALFORGE + SFT	9.9	26.1	23.7	4.4
Qwen3.5-9B				
Single-actor control	25.9	58.0	–	25.9
RESIDUALFORGE, selected design	32.1	63.0	34.4	28.2
RESIDUALFORGE + SFT	28.0	56.9	34.4	29.6
RESIDUALFORGE + OPD	35.8	66.7	37.4	30.4
RESIDUALFORGE + OPD + GRPO	36.0	67.1	36.6	31.1

Table 1: **Overall system results across three environments.** All scores use a 0–100 scale and are rounded to one decimal place; – denotes an unreported value. QCB denotes QwenClawBench. The blocks give external references, historical Qwen3-8B systems, and the principal Qwen3.5-9B systems. Metric definitions and reference sources appear in Appendices B and H.

Continuing with task-outcome rewards (GRPO) then raises TeamBench Pass from 30.37 to 31.11 and ClawEval pass^3 from 35.80 to 36.02, but QwenClawBench Score falls from 37.42 to 36.63. The full process therefore produces several useful endpoints, with the preferred training stage depending on the environment. This variation motivates the controlled studies below: the value of a policy update depends on the design and execution experience to which it is applied.

6 EFFECTS OF DEVELOPMENT CHOICES

The endpoint results combine decisions about what to test, how to organize execution, and what experience to learn from. We separate these decisions through controlled comparisons. Each study evaluates the complete system returned by its development condition, using its own frozen control and the common settings in Appendix F.

6.1 SYSTEM DESIGN CHANGES THE GAINS FROM TRAINING

Adaptive − fixed design	Before OPD	After OPD
ClawEval	−1.23	**+1.23**
QwenClawBench	−0.22	**+0.60**

Table 2: **Training reverses the ranking.** Differences in ClawEval pass^3 and QwenClawBench Score. Positive values favor adaptive design.

A system that performs well before training need not be the best starting point for learning. We compare fixed-design search, which optimizes prompts and skills in a two-role structure, with adaptive design, which can also change responsibilities, routes, communication, and system size. Both receive the same initial seed, development sources, and proposal budget. The selected designs are then frozen and trained with OPD. Table 2 compares their relative scores before and after training.

Training reverses the ranking on ClawEval and QwenClawBench. Adaptive design moves from 1.23 points behind fixed design to 1.23 ahead on ClawEval pass^3, and from 0.22 behind to 0.60 ahead on QwenClawBench. Its training gain exceeds that of fixed design by 2.46 and 0.82 points, respectively, using the displayed scores. On TeamBench, both untrained designs and fixed design + OPD reach 28.15, while adaptive design + OPD reaches 30.37. The complete scores and continuation results appear in Table 6. The ranking reversal links design choice to training outcome: choosing a design by its untrained score can select a different system from the one that performs best after training.

6.2 Execution evidence connects the development stages

System design and policy training both use execution evidence. We examine which parts of this evidence affect the final system by changing one source at a time: replacing mechanism-aware tasks with generic variants, removing revision history, omitting team handoffs from training context, or keeping downstream training data fixed. Table 3 shows how these changes affect ClawEval and QwenClawBench. Table 7 gives all three environments and both ClawEval metrics.

Change	ClawEval Δpass^3	QCB ΔScore
Generic tasks	-2.47	-1.46
No history	-1.23	-0.95
No handoffs	$\mathbf{-3.70}$	$\mathbf{-1.80}$
Fixed data	0.00	-0.48

Table 3: **Changes in final quality.** Differences from 35.80 ClawEval pass^3 and 37.42 QwenClawBench (QCB) Score.

The evidence used before training affects the final system. Generic task variants reduce ClawEval pass^3 by 2.47 points and QwenClawBench Score by 1.46, linking the quality of development experience to the system eventually returned. Removing revision history reduces the same metrics by 1.23 and 0.95 points. Current failures guide the next proposal, while the history records which previous changes repaired or regressed behavior. The feedback controls in Appendix D further separate these two sources of information.

The strongest drop occurs when training loses the context passed between roles. Omitting team handoffs lowers ClawEval pass^3 by 3.70 points and QwenClawBench Score by 1.80, with the deployed design held fixed. Keeping a fixed downstream corpus has a smaller effect: ClawEval pass^3 is unchanged, pass@3 falls by 1.24 points, and QwenClawBench Score falls by 0.48. These comparisons support retaining the execution context induced by the selected design and refreshing it as the policies change. On TeamBench, the full adaptive-design + OPD condition reaches 30.37 Pass. Removing revision history reduces it to 28.89, while generic task variants, omitted handoffs, and a fixed downstream corpus each reach 28.15.

6.3 Training across interfaces improves transfer

Interface	Zero-only	Pooled	Δ
ReAct *(unseen)*	33.33	35.80	$+2.47$
ZeroClaw	37.04	38.27	$+1.23$
OpenClaw	14.81	16.05	$+1.24$
Codex	16.05	22.22	$+6.17$

Table 4: **Transfer across interfaces.** ClawEval pass^3 after ZeroClaw-only or pooled OPD. ReAct is absent from the pool.

Training context also depends on the execution interface. We compare OPD using only ZeroClaw rollouts with OPD using an equal mixture of ZeroClaw, OpenClaw, and Codex episodes at the same total training budget. Each trained system is frozen and evaluated separately in all four interfaces. Table 4 marks ReAct as unseen because it is absent from the pooled training data. The tasks and grader are shared across these evaluations; Appendix F describes the fixed interface adapters.

Pooled training improves pass^3 in all four execution interfaces. The largest gain is in Codex, from 16.05 to 22.22. On ReAct, which is unseen during pooled training, pass^3 rises from 33.33 to 35.80. Pass@3 also improves in all four interfaces. The gain on ReAct shows that the benefit extends beyond the interfaces used to collect training experience. Absolute scores remain lower in OpenClaw and Codex, so the execution interface still affects the resulting performance. GRPO continuation raises pass^3 in three interfaces and leaves it unchanged in OpenClaw (Table 8).

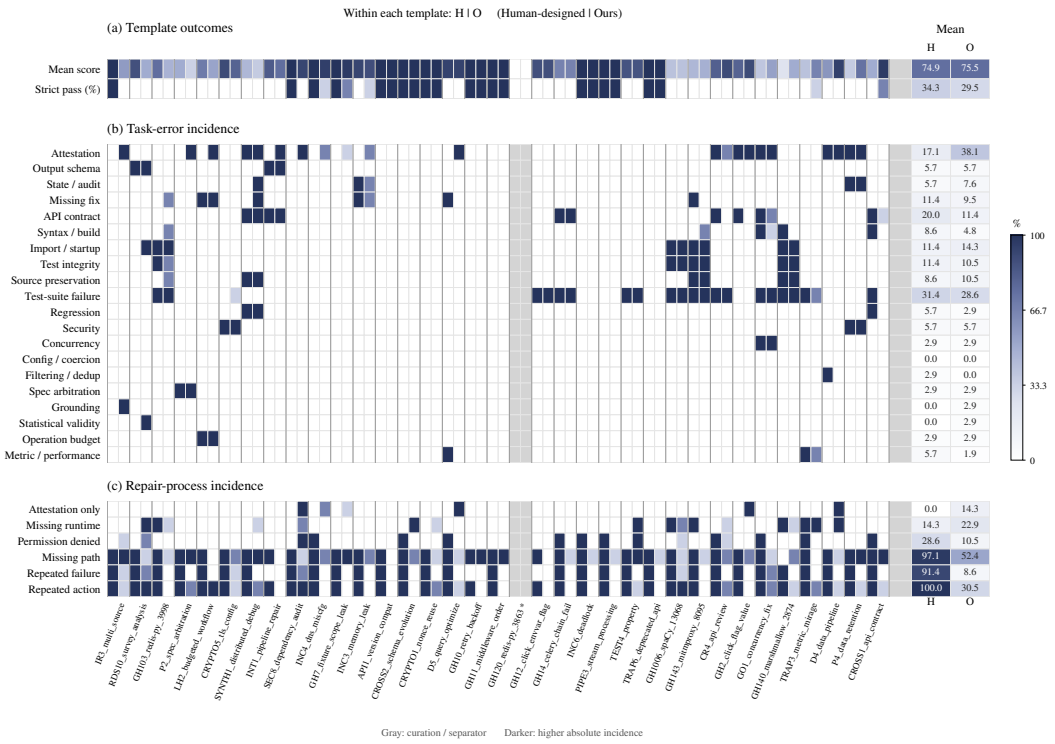


Figure 3: **Paired absolute outcome and failure profiles.** Each source template contributes adjacent Human (H) and Ours (O) columns. (a) Normalized score and strict pass rate. (b) Multi-label task-error incidence. (c) Observable repair-process incidence. Darker cells denote higher absolute values on a common 0–100 scale: higher is better in (a), whereas (b,c) measure category frequency, not quality. White denotes zero, not an unavailable observation. The rightmost pair gives means with equal source-template weighting; Ours averages repeated trials within each template. Gray marks curation-disabled error/process cells and the separator before the means. The curation-disabled template retains its recorded zero outcome in the aggregates.

Failure patterns relative to human-designed systems. Error analysis reveals a gap between partial progress and complete delivery. The selected system attains a similar mean score to the human-designed reference but lower strict completion, with final-attestation errors accounting for the largest incidence gap. Trace inspection identifies incomplete acceptance checks, destructive source edits, and ineffective recovery as recurring bottlenecks; fewer repeated tool calls alone do not ensure success. Appendix I provides the paired error profiles and case studies. The results are shown in Figure 3.

7 Conclusion

ResidualForge develops tool-using agent systems by connecting executable tasks, failure-guided design, and native policy learning. The experiments show why these decisions belong in one development process. Training can reverse the ranking of two system designs, removing execution context reduces the quality of the trained system, and broader rollout experience improves transfer across harnesses. System selection can also retain a single actor when larger candidates perform worse. These findings relate the quality of the returned agent to the choices that produce its design and training experience.

AI use statement

The framework uses language models to construct executable development tasks, propose system revisions, and generate or score training trajectories, as described in Section 4. We also used Chat-

GPT/Codex for feedback on methodological framing and interpretation of experimental results, literature search, manuscript organization and editing, title development, reference formatting, LaTeX preparation, figure design, and visualization code. Experimental scores are taken from the recorded evaluations described in Section 5 and Appendix B. The authors are responsible for the final text, claims, analyses, and artifacts.

REPRODUCIBILITY STATEMENT

Section 4 and Algorithm 1 describe task construction, system design, trajectory collection, and policy training. Appendix E specifies source partitioning and task lineage, and Appendix B defines the evaluation metrics, trial handling, and campaign-specific settings. Appendices D and F document the component interventions and controlled comparisons. Appendices C and G describe execution profiles and resource accounting. Historical configurations and published reference sources appear in Appendix H.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Yi Duan, Ying Liu, Zirui Tang, Haodong Chen, Jun Zhou, Yumou Liu, Bangrui Xu, Yukai Wu, Sidi Chen, Yuhan Zhou, et al. The last ai built by humans: Toward genuine recursive self-improvement. *arXiv preprint arXiv:2609.11873*, 2026.

Run-Ze Fan, Zihao Zhang, Simin Ma, Yebowen Hu, Shouju Wang, Kaiqiang Song, Fei Liu, Hamed Zamani, and Xiaoyang Wang. An empirical study of harness design for coding agents. *arXiv preprint arXiv:2609.20804*, 2026.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Zhiyuan He, Siwei Zhang, Zhiwen Zhou, Yuqing Yang, Yu Kang, Yuge Zhang, Luna K Qiu, Tin Yan Tsui, Jiahang Xu, and Chong Luo. Agent lightning v1. 0: Towards harnessed agentic rl. *arXiv preprint arXiv:2608.17528*, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Ryan Chin, Caiming Xiong, and Shafiq Joty. Mas-zero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*, 2025.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026a.

Zixuan Ke, Vaidehi Patil, Haizhou Shi, Yang Li, Ye Liu, Sarath Shekkizhar, Anurag Koul, Jiayu Wang, Xuan Phi Nguyen, Semih Yavuz, et al. Evoharnessbench: Can your agents keep pace with an evolving harness? *arXiv preprint arXiv:2609.04280*, 2026b.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Yubin Kim, Chanwoo Park, Taehan Kim, Eugene Park, Samuel Schmidgall, Salman Rahman, Chunjong Park, Cynthia Breazeal, Xin Liu, Hamid Palangi, et al. Teambench: Evaluating agent coordination under enforced role separation. *arXiv preprint arXiv:2605.07073*, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Y Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations*, volume 2026, pp. 50524–50570, 2026.

Tianyi Lin, Chuanyu Sun, Jingyi Zhang, Changxu Wei, Huanjin Yao, Shunyu Liu, Xikun Zhang, Liu Liu, and Jiaxing Huang. Openclaw-skill: Collective skill tree search for agentic large language models. *arXiv preprint arXiv:2606.16774*, 2026.

Haozhe Liu, Tian Ye, Sensen Gao, Qihang Cao, Yitong Li, Mingchen Zhuge, Duomin Wang, Ruihua Zhang, Ping Luo, Jiawang Bian, et al. Sol-pi: Recursively scaling auto-research loops for efficient agent harness. *arXiv preprint arXiv:2609.20519*, 2026.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, 2024.

Aneesh Pappu, Mirac Suzgun, Yongchan Kwon, Federico Bianchi, Batu El, Mykel J Kochenderfer, Hancheng Cao, and James Zou. Self-organizing agent teams learn to reason together. *arXiv preprint arXiv:2609.22682*, 2026.

Qwen Team and Alibaba Data. QwenClawBench: Real-user-distribution benchmark for openclaw agents, April 2026a. URL github.com/SKYLENAGE-AI/QwenClawBench.

Qwen Team and Alibaba Data. QwenClawBench: Real-user-distribution benchmark for openclaw agents, April 2026b. URL github.com/SKYLENAGE-AI/QwenClawBench.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Zhan Shi, Bing He, Yisi Sang, Hanqing Lu, and Benoit Dumoulin. A-evolve-training: Autonomous post-training of a 30b model. *arXiv preprint arXiv:2606.20657*, 2026.

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-tuning and prompt optimization: Two great steps that work better together. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10696–10710, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

Xiao Yu, Baolin Peng, Ruize Xu, Hao Zou, Qianhui Wu, Hao Cheng, Wenlin Yao, Nikhil Singh, Zhou Yu, and Jianfeng Gao. Openforgerl: Train harness-native agents in any environment. *arXiv preprint arXiv:2607.21557*, 2026.

Yirong Zeng, Shen You, Jinhang Feng, Yufei Liu, Xiao Ding, Yutai Hou, Hao Cong, Yuxian Wang, Wu Ning, Wang Xu, et al. Envcraft: Synthesizing executable environments in agentic rl for claw-like agent. *arXiv preprint arXiv:2609.05576*, 2026.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026b.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026c.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James Zou. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *Advances in Neural Information Processing Systems*, 38: 124475–124504, 2026.

Noah Ziems, Dilara Soylu, Lakshya A Agrawal, Isaac Miller, Liheng Lai, Chen Qian, Kaiqiang Song, Meng Jiang, Dan Klein, Matei Zaharia, et al. Composing policy gradients and prompt optimization for language model programs. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 197–213, 2026.

Environment	Metric	Before OPD	After OPD	Δ
ClawEval	pass^3	32.10	35.80	+3.70
ClawEval	pass@3	62.96	66.67	+3.70
QwenClawBench	Score	34.36	37.42	+3.07

Table 5: **OPD results in the main-table metrics.** Values use a 0–100 scale, and Δ is computed before rounding. ClawEval OPD runs 31 batches and is evaluated with a doubled task-time ceiling. QwenClawBench OPD runs 20 batches with the same 1,800-second cap, 65,536-token context, source split, and native grader before and after training.

A LIMITATIONS

The principal experiments use Qwen3.5-9B across three benchmark environments, and the transfer study covers four execution harnesses. Evaluating larger teams and additional model families would extend the scope of these findings.

The ClawEval OPD endpoint combines policy training with a larger task-time allowance. Resource settings and metric definitions are recorded in Appendix B.

Development operates within user-supplied goals, resources, permitted operations, and controller rules.

B EVALUATION DETAILS

The metric definitions and training details are given below. Historical Qwen3-8B configurations and published benchmark references are collected in Appendix H.

Reliability metrics. For binary ClawEval outcomes b_{ij} on N tasks and three trials,

$$\text{pass}^3 = \frac{100}{N}\sum_i \mathbf{1}\left[\sum_{j=1}^{3} b_{ij} = 3\right], \qquad \text{pass@3} = \frac{100}{N}\sum_i \mathbf{1}\left[\sum_{j=1}^{3} b_{ij} > 0\right]. \tag{4}$$

The original binary metrics are also given in Equation 6. An infrastructure retry re-executes the affected trial and counts once. For QwenClawBench, with m_i executions of task i and fractional native scores s_{ij}, Score is $100N^{-1}\sum_i m_i^{-1}\sum_{j=1}^{m_i} s_{ij}$. Equal trial counts reduce this to the mean of all task–trial scores. OpenForgeRL reports this quantity under the label pass@1 (Qwen Team & Alibaba Data, 2026b; Yu et al., 2026). TeamBench averages local strict binary verdicts; published attestation-promoted scores use a different evaluation rule.

Paired transitions. Paired transitions count repaired and regressed tasks using Equation 1. The ClawEval single-actor, selected-design, and OPD conditions retain their recorded execution allowances. The OPD task-time ceiling is twice the selected design's ceiling.

Model, routing, and resources. In the archived ClawEval OPD comparison, both systems use Qwen3.5-9B and the same frozen pair specification (identical canonical content hash), but their task-time ceilings differ. Processed-token means include repeated prompt prefixes, and wall time is the per-run elapsed time including execution and serving. Task timeouts count as unsuccessful outcomes.

Development and training details. On the ClawEval development sources, which also supply training experience, the doubled-time evaluation gives rubric means of 61.53 before and 64.92 after OPD. The QwenClawBench OPD campaign uses the native complete-evidence scorer; its 3.07-point gain from unrounded values corresponds to an 8.93% relative increase. The OPD role-update map contains the action-owner policy.

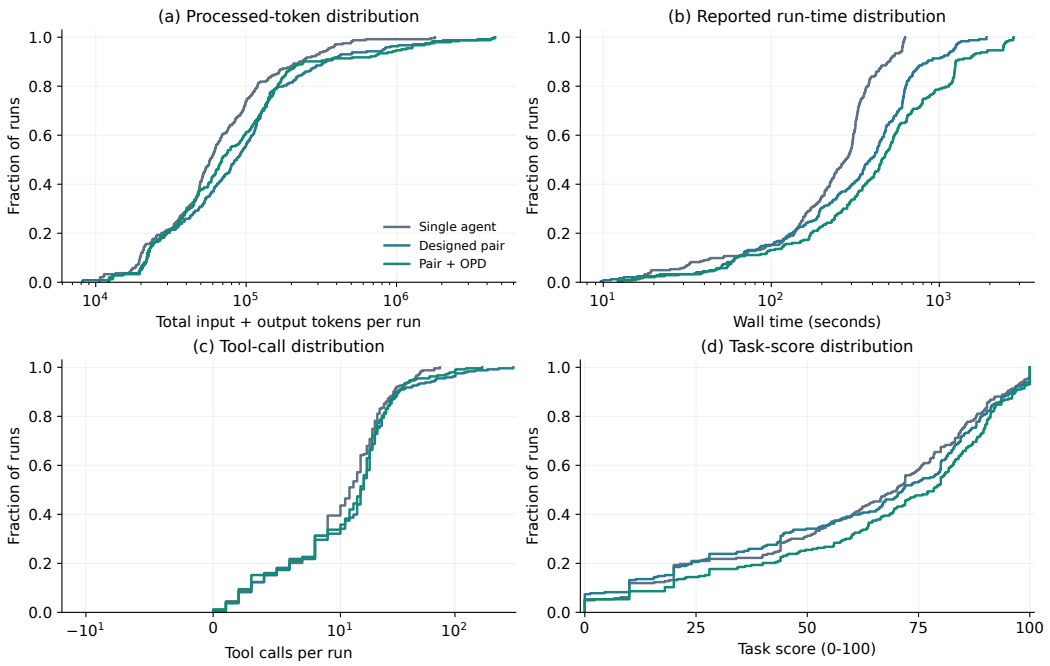


Figure 4: **Execution profiles.** Empirical cumulative distributions of processed tokens, wall time, tool calls, and task score for the single agent, the designed pair, and the OPD model. In panel (d), the vertical axis is the fraction of runs scoring at or below the horizontal value.

B.1 Teacher-trajectory SFT

ClawEval SFT reaches 27.99 pass^3 and 56.85 pass@3, as reported in Table 1. QwenClawBench teacher-SFT reaches 34.42 against 34.36 for its control. Historical Qwen3-8B SFT follows its recorded benchmark-specific recipes (Appendix H).

B.2 Earlier QwenClawBench configuration and TeamBench selection

An earlier QwenClawBench configuration, with a different scorer and limits, gave untrained rubric means of 40.19, 41.18, and 44.90 for the single-, two-, and three-actor systems. The OPD campaign reruns its own frozen control at 34.36 under the native complete-evidence scorer, so OPD effects are computed within that campaign.

On TeamBench, the parent's development rubric score is 77.84, and three successive candidate cycles reach 75.10, 71.05, and 75.66. None exceeds the parent, so the designer returns the parent, whose Pass is 28.15.

C Execution audit

OPD uses fewer tool calls but more time and tokens. After OPD, mean tool calls fall from 21.64 to 18.36 per run, while mean wall time rises from 440.82 to 639.96 seconds and mean processed tokens per run (input plus output, summed over model calls) rise from 207.14k to 220.54k under the doubled time ceiling. Figure 4 shows the full distributions.

D System-level study details

Table 6 gives the full design–training grid and the supplementary controls. Tables 7 and 8 retain all outcomes underlying the compact main-text tables. The comparison definitions below connect each intervention to the question it tests.

Study	Configuration	ClawEval pass^3	ClawEval pass@3	QCB Score	TeamBench Pass
Design freedom × training					
F	No training	33.33	64.20	34.58	28.15
D	No training	32.10	62.96	34.36	28.15
F	OPD	34.57	64.20	36.82	28.15
D	OPD	35.80	66.67	37.42	30.37
F	OPD + more OPD	34.57	65.43	36.95	27.41
D	OPD + more OPD	35.80	65.43	37.20	28.15
F	OPD + GRPO	35.80	65.43	37.13	27.41
D	OPD + GRPO	36.02	67.08	36.63	31.11
Additional diagnostic controls not repeated in Table 7					
Panel	Random source panel	32.10	61.73	35.44	27.41
Growth	Parent + matched extra compute	35.80	66.67	37.28	28.89
Growth	Generic always-active helper	33.33	62.96	35.88	26.67
Growth	Always-active specialist	34.57	65.43	36.76	27.41
Feedback	Scalar feedback, no history	32.10	61.73	35.71	27.41
Feedback	Scalar feedback + history	33.33	64.20	36.21	28.15

Table 6: **Design–training grid and supplementary controls.** F changes prompts and skills within a fixed structure; D may also change responsibilities, routes, communication, and team size. Each selected design is frozen before training. Continued OPD and GRPO start from the same OPD checkpoint within each design. The lower block reports the additional panel, role-allocation, and feedback controls. Appendix F specifies the common settings.

Development condition	ClawEval pass^3	ClawEval pass@3	QCB Score	TeamBench Pass
Fixed-design search + OPD	34.57	64.20	36.82	28.15
Adaptive design + OPD	35.80	66.67	37.42	30.37
Generic task variants	33.33	64.20	35.96	28.15
Current feedback without revision history	34.57	65.43	36.47	28.89
Role-local histories without team handoffs	32.10	61.73	35.62	28.15
Fixed downstream training corpus	35.80	65.43	36.94	28.15

Table 7: **Development and training choices.** Each intervention changes one factor relative to adaptive design + OPD under the study's common source, teacher, proposal, training, and execution settings. The fixed-design-plus-OPD reference is reported in Table 6. QCB denotes QwenClawBench.

For environment-specific quality Q, the design–training interaction is

$$I_{\mathrm{OPD}} = \big(Q_{D,\mathrm{OPD}} - Q_{D,0}\big) - \big(Q_{F,\mathrm{OPD}} - Q_{F,0}\big), \tag{5}$$

where D and F denote adaptive and fixed design. The displayed values give 2.46 points for ClawEval pass^3, 3.71 for pass@3, 0.82 for QwenClawBench Score, and 2.22 for TeamBench Pass.

D.1 Development experience

The experience study compares random source sampling, generic semantic variants, and mechanism-aware task construction under the same panel size and proposal/training budgets. Each arm retains source lineage and runs its own design search and OPD. Generic variants appear in Table 7; random source sampling appears in Table 6. Their ClawEval pass^3/pass@3 values are 33.33/64.20 and 32.10/61.73, respectively, compared with 35.80/66.67 for mechanism-aware construction. The corresponding QwenClawBench scores are 35.96, 35.44, and 37.42.

Training condition	Evaluation harness			
	ReAct	ZeroClaw	OpenClaw	Codex
Reliability across three trials: pass^3				
No training	32.10	34.57	13.58	14.81
OPD: ZeroClaw only	33.33	37.04	14.81	16.05
OPD: three-harness pool	35.80	38.27	16.05	22.22
OPD + GRPO: three-harness pool	36.02	39.51	16.05	24.69
At least one successful trial: pass@3				
No training	62.96	65.43	33.33	35.80
OPD: ZeroClaw only	64.20	67.90	34.57	37.04
OPD: three-harness pool	66.67	69.14	37.04	44.44
OPD + GRPO: three-harness pool	67.08	69.14	38.27	48.15

Table 8: **Cross-harness evaluation on ClawEval.** Each training condition fixes a Qwen3.5-9B system and checkpoint. Every column is a separate evaluation under the common protocol in Appendix F. The pool contains ZeroClaw, OpenClaw, and Codex episodes; ReAct is unseen during pooled training. The two panels preserve both reliability and at-least-one-success outcomes.

D.2 Role allocation

Starting from the same parent, the role-allocation comparison allocates additional computation to the existing actor, adds a generic helper, uses an always-active specialist, or conditionally activates that specialist. The generic and specialized helpers share placement, information access, and return protocol. The always-active and conditional specialists share their responsibility and differ in activation. All conditions use one team-level resource envelope. These controls separate additional inference, differentiated responsibility, and selective activation.

Under these controls, additional computation for the parent matches the conditional system on ClawEval, reaches 37.28 versus 37.42 on QwenClawBench, and reaches 28.89 versus 30.37 on TeamBench (Table 6). Both a generic helper and an always-active specialist perform worse than the conditional system on ClawEval and QwenClawBench. This makes extra parent computation a useful comparator when deciding whether a task benefits from another role.

D.3 Feedback and revision history

The feedback comparison crosses scalar or rich current feedback with the absence or presence of an archive of repairs and regressions. Rich feedback contains the delivered role inputs and outputs, tool returns, permitted item-level checks, and paired traces. History contains accepted and rejected changes, positive and negative deltas, and successful behavior to preserve. All arms fork the same saved parent and pre-intervention archive. Information masks prevent the scalar-history arm from recovering excluded current traces through older records.

On QwenClawBench, scalar feedback reaches 35.71 without history and 36.21 with history. Rich current feedback without history reaches 36.47, and the complete condition reaches 37.42. The displayed feedback–history interaction is $(37.42-36.47)-(36.21-35.71) = 0.45$ points. ClawEval and TeamBench outcomes remain visible in Tables 7 and 6.

D.4 Training context and continuation

After system selection, prompts, skills, routing, and tool schemas are fixed. The OPD interventions vary team-conditioned versus role-local histories and fresh rollouts versus a fixed corpus. Each arm uses executed observations, the same teacher and student initialization, and a common token/update budget. Recorded outputs of the target role supply its training targets.

The GRPO continuation and more-OPD control fork the same OPD checkpoint within each design. GRPO groups independent native executions by development task and uses their executable outcome rewards. The comparison fixes the additional accelerator-time and student-token ceilings.

1. Receive goal, actors, native harness, source partitions, search limits, training recipe, and resources.
2. Construct and audit executable development tasks; freeze the selection panel for each comparison block.
3. Start from a single actor or validated parent; diagnose failures from role traces and permitted outcome checks.
4. Jointly revise responsibilities, communication, prompts/skills, and conditional growth using explicit target cases and records of repairs and regressions.
5. Retain the best representative-panel proposal, evaluate it on the full development panel, and apply the configured promotion/stopping rule; preserve the validated parent on rejection.
6. Freeze the selected specification and bind trainable roles, native interface, source split, and initial checkpoints.
7. Collect teacher demonstrations for SFT or current-student executions for online training; retain actual role histories and tool observations.
8. Execute the configured SFT, OPD, or reward-based backend. For online updates, collect fresh rollouts under the updated policies; persist checkpoints and role mappings.
9. Freeze the endpoint; evaluate protected tasks and export the decision and resource ledger.

Algorithm 1: AI-directed system construction and configurable native-policy training

Teacher usage, reward variation, and skipped zero-variance groups are separate resource and training diagnostics.

The adaptive-design OPD and OPD + GRPO endpoints agree with Table 1. GRPO reaches 36.02/67.08 on ClawEval, compared with 35.80/65.43 after continued OPD. Its QwenClawBench Score is lower, 36.63 versus 37.20, while TeamBench Pass is higher, 31.11 versus 28.15. Thus, outcome-reward continuation does not dominate additional distillation in every environment.

E Source partitions and executable experience

Original sources are split before any derivative is generated, and variants, repaired trajectories, and derived assets inherit their source's partition. The pipeline records the purposes of adaptive diagnosis, within-block selection, gradient training, checkpoint validation, offline audit, and protected testing. The campaign contract records which stages share development sources and which use separate partitions. Protected evaluation sources are reserved for final evaluation.

The generator's mechanism checklist includes independent file discovery, cross-file dependencies, causal action order, state-changing operations, artifact type, and evidence of real side effects. Public task text contains the goal and never the private grader answer or a required solution path. Topic diversity and mechanism diversity are separate statistics. Quality reports retain rejected tasks, uncovered mechanisms, normalized prompt duplication, workspace-path diversity, and grader-family diversity.

F Matched study protocols

Development experience. Each panel arm runs its own design search and training under the same proposal, rollout, and training ceilings, and source families stay separated across training, selection, and audit.

Additional roles. Control A spends the extra budget on the parent; B adds a generic helper; C gives that helper a targeted specialization with the same information access and return location; D adds a routing predicate to the same specialist. All four share one total token/time envelope. A controls for additional inference, C–B isolates differentiated responsibility, and D–C isolates selective activation.

Current feedback and revision history. The four feedback/history arms fork a common parent, checkpoint, and archive. The history archive holds earlier positive and negative changes; current hidden traces, target-case evidence, and source-sibling evidence are masked consistently across arms. Each decision retains task identities and structural/prompt diffs.

OPD data and GRPO. Teacher and student share the native role prompt, tool schema, reasoning serialization, and sampled-token identity. Only the current role's assistant tokens receive its update; tool returns and other roles' outputs are conditioning context. A changed upstream policy shifts the downstream input distribution, so refresh is a separate intervention whose rollout cost is counted. The GRPO stage collects multiple executions per training source and computes group-relative outcome advantages, with group size, update count, total generated tokens, and reward definition fixed before the comparison with continued OPD. Failed executions count toward rewards unless they are confirmed infrastructure failures.

Design–training factorial. F and D share the initial two-role seed, proposer model, feedback, candidate-count ceiling, development sources, and team-level evaluation limits. F searches role prompts and skills within the declared responsibilities and routes; D also searches those structural decisions and team size. Generated specifications are checked for compliance with each treatment, and the selected F and D specifications are frozen before their weight-update conditions. Continued OPD and GRPO fork the same trained parent within each design arm with identical accelerator-time and student-token ceilings, and teacher tokens are tracked as an additional resource. Within each training stage, F and D share actor context and task-time limits.

G Development and training cost accounting

The campaign specification fixes the implementation, user goal, actor checkpoint, source split, harness and grader versions, resources, and allowed operations. Within that contract, the designer chooses development experience and system proposals, and the fixed controller performs execution, bounded retries, acceptance, stopping, and checkpoint preservation. The actor team produces native actions and deliverables, which the grader verifies.

The cost ledger separates synthesis, designer, teacher, actor, grading, replay, unsuccessful-attempt, and gradient costs. Final system quality and development and training costs are reported separately.

H Published references and historical systems

Table 1 reports the external reference systems, historical local Qwen3-8B configurations, and principal Qwen3.5-9B results together. This appendix records the sources, metrics, and historical settings behind those entries. OpenForge-Claw is trained from Qwen3-30B-A3B-Thinking. Published TeamBench entries use attestation promotion; local historical entries use strict verdicts on development instances.

The historical selected systems improve ClawEval pass^3/pass@3 from 3.11/18.63 to 4.97/26.71 and QwenClawBench Score from 16.40 to 24.27. SFT raises ClawEval pass^3 to 9.94 while pass@3 changes to 26.09 and QwenClawBench Score to 23.71. TeamBench development Pass rises from 0.74 to 4.44 after design and remains 4.44 after SFT.

H.1 Published cross-harness references

Table 9 gives OpenForgeRL's cross-harness results in their original binary metrics. For binary outcomes b_{ij} over three trials, these metrics are

$$\widehat{\text{pass@1}} = \frac{1}{3N}\sum_{i,j} b_{ij}, \quad \widehat{\text{pass@3}} = \frac{1}{N}\sum_{i} \mathbf{1}[\sum_{j} b_{ij} > 0], \quad \widehat{\text{pass}^3} = \frac{1}{N}\sum_{i} \mathbf{1}[\sum_{j} b_{ij} = 3]. \quad (6)$$

System	Training sources	pass@1 (%)				pass@3 (%)			
		ReAct	Zero	Open	Codex	ReAct	Zero	Open	Codex
Published Qwen3-30B-A3B reference: full ClawEval general-task track									
Base	Base	26.10	32.50	11.40	12.20	39.80	44.70	19.30	18.60
OpenForge SFT	Mixed harnesses	36.20	44.50	16.70	21.10	52.10	66.50	24.20	35.40
OpenForge SFT+RL	Native training	45.10	48.50	20.90	32.50	55.90	67.10	27.80	51.50

Table 9: **Published cross-harness references in their original metrics.** OpenForgeRL's Qwen3-30B-A3B results on the full ClawEval general-task track (Yu et al., 2026). Zero/Open denote ZeroClaw/OpenClaw.

H.2 Public-result sources

ClawEval and QwenClawBench reference values come from Yu et al. (2026), Table 2, with additions checked on September 23, 2026. ClawEval Sonnet 4.6 (68.30 pass^3, 81.40 pass@3) and Gemini 3 Flash (48.40, 67.10) come from the general-task columns of Ye et al. (2026), Table 3. QwenClawBench Sonnet 4.6 (60.50) comes from the official v1.1 leaderboard (Qwen Team & Alibaba Data, 2026a); Qwen3-8B (11.50) comes from the untrained baseline in Lin et al. (2026), Table 1. TeamBench full-team references come from the official leaderboard (Kim et al., 2026). Table 1 displays values to one decimal place; the detailed tables retain their reported precision.

H.3 Comparison records

Appendix B.1 reports SFT results, and Appendix F documents the controlled studies.

I FAILURE PATTERNS RELATIVE TO HUMAN-DESIGNED SYSTEMS

Both arms in this diagnostic comparison use base Qwen3.5-9B without post-training. Human denotes a human-authored, model-executed Planner/Executor/Verifier system, not human task solvers; Ours is the selected owner-only system. The paired absolute values distinguish shared failures from categories absent in both systems.

Similar partial credit does not imply complete delivery. Our system reaches a mean score of 75.46 versus 74.87 for the human-designed reference, but a lower strict pass rate: 29.52% versus 34.29% (Figure 3a). The differences are +0.59 score points and -4.76 percentage points, respectively. Final-attestation errors show the largest task-error incidence excess (+21.0 points); attestation is the only recorded error category in 14.29% of our runs, versus none for Human. The inspected D4 trace illustrates the distinction: the actor completes the data processing, writes `status=complete` and `verified=true`, and reads the file back, whereas acceptance requires `verdict=pass`. Readback checks persistence, not satisfaction of the completion contract.

Aggregate similarity hides task-specific gains and regressions. The paired outcome and task-error blocks expose heterogeneous outcomes (Figure 3a,b). Our system improves on data-pipeline and API-contract templates, consistent with lower API-contract and syntax/build error incidence (-8.6 and -3.8 points). However, import/startup and source-preservation errors are more frequent (+2.9 and +1.9 points). In the inspected GH103 traces, a local repair replaces substantial source modules with short or empty versions, losing surrounding functionality. IR3 exposes a different limitation: authoritative documents are initially absent in our workspace, after which the actor creates its own corpus and answers from it. Human receives authoritative material. That gap combines an asset-provisioning difference with an evidence-integrity failure, rather than isolating the effect of the role design.

Fewer repeated calls are not evidence of better recovery. Repeated-failure and repeated-action incidence are lower for Ours (-82.9 and -69.5 points; Figure 3c), yet strict completion is lower. These indicators count runs containing recurring identical tool calls or failures; they are not mutually exclusive causes, and repeated calls can serve legitimate verification. In the inspected SEC8 trace, failed dependency-installation attempts nevertheless recur until timeout, leaving the completion declaration unresolved. The diagnostic priorities are therefore contract-aware final checks, preservation of existing artifacts, and recovery that reaches a verifiable stopping condition, rather than minimizing tool repetition alone.

False acceptance is omitted because matched final-attestation artifacts are unavailable for Ours; missing evidence is not treated as zero. Budget-stop indicators are omitted because their definitions differ across the two arms.