ANCHOROPT: STRUCTURED RUNTIME OPTIMIZATION IN AGENT HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent harnesses govern consequential runtime decisions around language models, including tool invocation, persistent-state modification, and error recovery. Optimizing these decisions jointly is challenging: intervention placement determines what can be observed and executed, signals and policies must be designed together, and local modifications interact through downstream trajectories. We introduce AnchorOpt, a structured runtime optimization framework that decomposes this coupled harness-design problem into tractable local decision problems. First, backward localization identifies consequential decisions from residual failures, while runtime boundaries constrain observable signals and feasible interventions. Second, the optimizer searches for local policies under available signals and expands the signal representation when search stalls. Third, accepted controllers are installed sequentially, with residuals re-mined and candidates evaluated against the evolving incumbent by their downstream task value. The resulting controllers have explicit runtime activation and execution semantics. Experiments on BFCL Agent Memory and AppWorld demonstrate improved task performance and complementarity with global prompt optimization. Additional studies examine repeated-execution reliability and the trade-off between constraint mitigation and intervention cost.

1 INTRODUCTION

LLM agents repeatedly make consequential runtime decisions: whether to invoke tools, modify persistent state, retrieve information, recover from errors, or terminate. These decisions are interdependent: an inappropriate memory write may cause a retrieval failure several steps later, while a locally sensible recovery action may degrade subsequent performance. Improving agent performance and reliability therefore requires optimizing not only the language model, but also the execution system governing its interactions with tools, state, and the environment.

Recent approaches have demonstrated the value of optimizing prompts and compound LM programs (Khattab et al., 2024; Agrawal et al., 2025), workflows (Zhang et al., 2025a), and increasingly the agent harness itself (Lee et al., 2026; Zhang et al., 2026; Lin et al., 2026). Yet harness optimization presents a challenging sequential decision problem. Agent execution may involve stochastic generation, large action spaces, and long horizons, with earlier actions changing the states and decisions encountered later. Failures may become observable only long after the decisions responsible for them. Moreover, the information available and corrective actions feasible at any point depend on when intervention occurs: a proposed tool call can be modified before execution, whereas its realized result is observable only afterward. Optimizing a harness thus requires jointly determining *where* to intervene, *what* to observe, *how* to act, and which interventions to deploy together. The resulting search over interacting runtime policies and their possible continuations can become computationally prohibitive.

We introduce AnchorOpt[1], a framework that makes harness optimization more tractable by representing runtime control as constrained local decision problems. As illustrated in Figure 1, each decision boundary restricts both observable information and feasible interventions, eliminating invalid signal–action combinations before evaluation. The resulting controllers implement executable conditional control flow rather than merely modifying the host model's instructions: the runtime evaluates locally observable conditions, dispatches the prescribed intervention when applicable, and otherwise preserves native execution.

[1]Anonymized code: `https://anonymous.4open.science/r/anchoropt-7644`

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

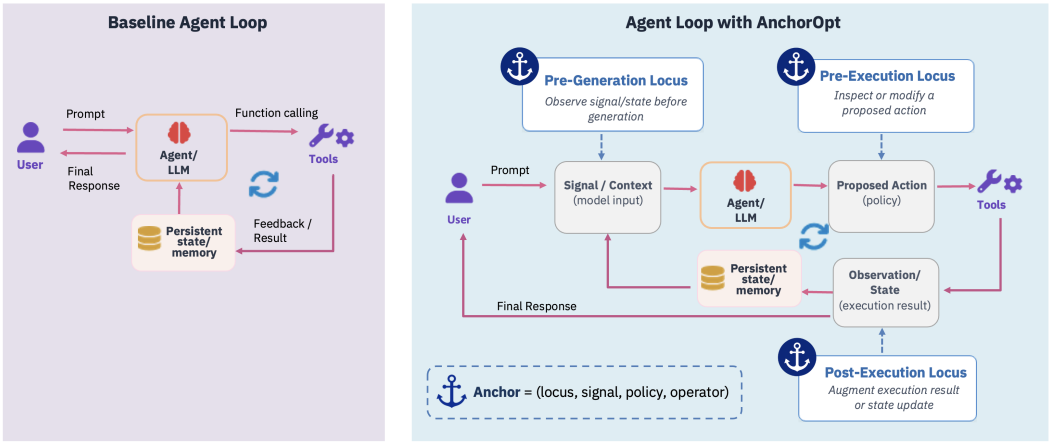


Figure 1: Baseline agent execution (left) and AnchorOpt runtime control (right). Each decision locus constrains the observable signals and feasible interventions of its local controller.

Within this constrained space, AnchorOpt improves tractability in two additional ways. First, it alternates between searching for local intervention policies under a fixed signal representation and expanding that representation only when no further improvement can be found. Second, it constructs the controller portfolio sequentially rather than optimizing all interacting controllers jointly. Each accepted controller updates the incumbent harness, after which residual failures are re-mined to guide subsequent search. Together, these design choices replace a large, coupled harness-design problem with a sequence of smaller, empirically evaluated optimization problems, while retaining end-to-end task performance as the criterion for accepting interventions.

A central component is backward localization: the point where a failure becomes visible need not be the most effective place to intervene. AnchorOpt traces residual failures to earlier consequential decisions and searches for locally observable conditions under which feasible interventions improve downstream outcomes. In this way, the framework provides structured, decision-local control without requiring an explicit model of the agent's transition dynamics or exhaustive search over possible harnesses. Our contributions are:

- **A structured formulation of runtime harness optimization.** We represent harness adaptation as optimization over boundary-conditioned local decision spaces, in which observability and feasible actions are jointly constrained by runtime location. This formulation prunes infeasible interventions and turns an open-ended harness-design problem into a collection of mechanically valid local search spaces.
- **A decomposed optimization algorithm.** We combine backward failure localization, alternating signal–policy search, and sequential portfolio construction against a moving incumbent. This avoids unrestricted joint search over signal representations, local policies, and interacting controllers, while retaining empirical downstream value as the criterion for accepting improvements.
- **An empirical investigation of effectiveness, structure, and portability.** Across tool-use and memory-intensive agent benchmarks, we evaluate the resulting controller portfolios and investigate how intervention placement, observability, and feasible actions affect downstream performance. We further examine the benefits of decision-local control for intervention frequency and execution cost, alongside the portability of structured runtime control across host models and agent environments.

2 Related Work

Optimization around fixed language models. TextGrad (Yuksekgonul et al., 2024), DSPy (Khattab et al., 2024), and GEPA (Agrawal et al., 2025) optimize instructions or compound LM programs, while ACE (Zhang et al., 2025b) adapts evolving contexts. Automated agent design further searches over workflows and agent architectures, including AFlow (Zhang et al., 2025a) and ADAS (Hu

et al., 2025), alongside related approaches (Ye et al., 2026a; Shang et al., 2025; Yuan et al., 2025). Recent work extends optimization to the execution harness itself. Self-Harness (Zhang et al., 2026) mines failures, proposes harness modifications, and regression-tests changes. Other approaches search executable harness programs or guide harness evolution through failure attribution, persistent refinement, retrospective trajectories, and structured patching (Lee et al., 2026; Lin et al., 2026; Karten et al., 2026; Pan et al., 2026; Park et al., 2026). AnchorOpt complements these approaches by structuring the runtime-control search space around boundary-conditioned observation and action sets, alternating signal–policy search, and sequential controller selection.

Runtime control and failure localization. Reflexion (Shinn et al., 2023) uses failure feedback to guide subsequent attempts. Related work introduces runtime enforcement and conditional guardrails (Wang et al., 2025; Chen & Cong, 2025), while execution-level evaluation and trace analysis investigate agent risks, failure taxonomies, intermediate-step assessment, and error localization (Ruan et al., 2024; Cemri et al., 2025; Deshpande et al., 2025; Zhuge et al., 2024). AnchorOpt distinguishes failure diagnosis from intervention optimization: the observed failure need not identify the most effective decision boundary for correction, and a mechanically valid repair need not improve downstream task performance.

Sequential optimization and credit assignment. Backward localization relates to hindsight and counterfactual credit assignment (Foerster et al., 2018; Harutyunyan et al., 2019; Mesnard et al., 2021) and trajectory-level agent learning (Luo et al., 2025). Recent work also advocates Bayesian decision-making at the agent orchestration layer (Papamarkou et al., 2026). Building on a decision-centric perspective (Tsiourvas et al., 2025; Sun, 2026), AnchorOpt optimizes feasible runtime controllers around a fixed model through empirical downstream evaluation, without requiring an explicit environment model.

3 Structured Runtime Optimization

3.1 From Global Harness Optimization to Local Decisions

Let H_0 denote a native harness for a fixed host model. An execution generates a trajectory τ with realized task reward $R(\tau)$. A portfolio $\mathcal{P} = (A_1, \ldots, A_K)$ is an ordered collection of local controllers installed into the harness, with $H_0 \oplus \mathcal{P}$ denoting the resulting execution system. Let $\mathfrak{P}(H_0)$ denote the set of mechanically admissible controller portfolios. The global harness-design problem is

$$\max_{\mathcal{P} \in \mathfrak{P}(H_0)} J(\mathcal{P}), \qquad J(\mathcal{P}) = \mathbb{E}_{\tau \sim H_0 \oplus \mathcal{P}}[R(\tau)], \tag{1}$$

subject to applicable safety and resource constraints. The expectation is over the task distribution and stochastic execution under the installed portfolio. Our experiments primarily use task success as reward; latency, execution cost, and intervention frequency can also enter the objective (Appendix A.4).

Boundaries define feasible local decision problems. Let H denote the current harness, including previously installed controllers, and let $\mathcal{L}(H)$ be its exposed runtime decision boundaries. At boundary $\ell \in \mathcal{L}(H)$, let $\mathcal{O}_\ell^H$ denote the set of possible local observations, containing only information available when control is exercised. For a realized observation $o \in \mathcal{O}_\ell^H$, let $\mathcal{U}_\ell^H(o)$ denote the set of executable runtime actions. Thus, selecting a boundary jointly restricts what a controller can observe and what it can still change. Table 1 illustrates three generic boundaries.

Boundary-conditioned controllers. An *anchor* is a local controller

$$A = (\ell, \phi, \mu), \qquad z = \phi(o), \quad u = \mu(z) \in \mathcal{U}_\ell^H(o), \tag{2}$$

where ℓ specifies the runtime boundary, ϕ extracts a decision-relevant representation z from the local observation, and μ maps that representation to an executable runtime action. The representation may include both triggering conditions and typed information needed to parameterize the intervention.

Typical operator types preserve native execution (NoOp), reprompt the model (Reprompt), suppress a pending call (Suppress), substitute a pending call (Substitute), or transform intermediate

Boundary	Available information	Possible control semantics
Pre-generation	Current context and persistent state; no proposed call yet	Change/augment context or request a new generation; cannot inspect a future proposed call
Post-generation / pre-execution	Proposed response or tool call and its arguments, plus current state	Pass through, request reconsideration, suppress or replace a pending call when executable
Post-execution	Realized tool result/error and updated state	Route recovery or transform the observation/state for future decisions; cannot undo completed execution merely by suppressing it

Table 1: Runtime boundaries constrain observable information and feasible interventions.

information (TRANSFORM). These denote executed behavior rather than instructions or telemetry labels. Their feasibility depends on the boundary and realized local state; we assume NOOP is always feasible.

The constrained controller space. Let Φ_ℓ denote the currently available signal extractors at boundary ℓ. Collect these families as $\Phi = \{\Phi_\ell\}_{\ell \in \mathcal{L}(H)}$. The feasible local controller space is

$$\mathcal{C}_\ell(H, \Phi) = \left\{ (\ell, \phi, \mu) : \ \phi \in \Phi_\ell, \quad \mu(\phi(o)) \in \mathcal{U}_\ell^H(o), \quad \forall o \in \mathcal{O}_\ell^H \ \right\}. \tag{3}$$

This boundary-conditioned representation restricts search to feasible local controllers, eliminating invalid observation–action combinations before empirical evaluation.

The remaining optimization challenge. Feasibility pruning alone does not solve Eq. (1). Even within the constrained space, signal representations and intervention policies are coupled, controllers interact through downstream execution, and their effects must be evaluated over stochastic trajectories. Jointly optimizing these choices and the resulting portfolio remains computationally challenging. AnchorOpt therefore combines boundary-conditioned search with alternating signal–policy optimization and sequential portfolio improvement, as developed next.

3.2 DECOMPOSING HARNESS SEARCH

The formulation in Eq. (1) couples intervention placement, signal representation, local policy design, and controller interactions. AnchorOpt makes this search more tractable through three complementary decompositions.

1. Boundary-conditioned search and backward localization. AnchorOpt restricts search to a small operator vocabulary at three types of runtime boundary: pre-generation, post-generation/pre-execution, and post-execution. Each boundary jointly constrains observable information and executable actions, eliminating infeasible signal–action combinations before evaluation. Backward localization further focuses search on consequential decisions implicated by residual failures, rather than every possible intervention point. Together, these restrictions replace open-ended harness modifications with smaller, mechanically feasible local decision problems. Each candidate specifies executable conditional control, allowing its activation, downstream value, and interactions with the incumbent portfolio to be evaluated explicitly.

2. Alternating signal and policy search. Even at a fixed boundary, controller design couples the representation ϕ with the response policy μ. Searching over all representations and policies jointly can require evaluating many combinations.

AnchorOpt instead searches feasible policies using the currently available signal families Φ. If completed, budgeted search yields no accepted controller while consequential residuals remain, it holds the portfolio fixed and expands Φ. This separates policy improvement from representational expansion, rather than repeatedly restarting an unrestricted joint search.

Changing the boundary and expanding Φ are distinct: the former changes the feasible local decision problem, whereas the latter changes what can be represented within it. We use "block-coordinate-style" to describe this discrete alternating search, not continuous coordinate descent.

3. Sequential portfolio optimization and counterfactual evaluation. Local controllers interact through trajectories and persistent state, so their value depends on which other controllers are installed. Rather than searching jointly over controller combinations, AnchorOpt evaluates candidates against the current incumbent and accepts one validated controller at a time.

For incumbent $\mathcal{P}_k$, the empirical marginal value of candidate A is

$$\widehat{\Delta}_d(A \mid \mathcal{P}_k) = \widehat{J}_d(\mathcal{P}_k \oplus A) - \widehat{J}_d(\mathcal{P}_k), \qquad d \in \{\text{train}, \text{dev}\}, \tag{4}$$

where $\widehat{J}_d$ denotes the empirical estimate of J on split d, and k indexes accepted portfolio updates.

Backward localization identifies where an intervention might help; counterfactual forward evaluation determines whether it improves downstream task outcomes. A candidate is promoted only after satisfying the prescribed performance and validity checks, rather than on failure diagnosis or local mechanism engagement alone.

Following acceptance, trajectories under $\mathcal{P}_{k+1}$ are regenerated and residuals are re-mined. This replaces exhaustive joint portfolio search with a sequence of marginal optimization problems, without claiming to solve Eq. (1) globally.

Optimization procedure. The resulting loop alternates between candidate-policy search under fixed signals and signal expansion when no candidate is accepted; every accepted controller updates the incumbent and triggers residual re-mining. Let $\mathcal{L}_c$ denote the consequential boundaries localized from current failures. The full pseudocode, including trajectory and candidate-set notation, is in Appendix A.3 (Algorithm 1).

3.3 Autonomous Learning Architecture and Deployment

AnchorOpt separates a reusable optimization core from an environment-specific runtime adapter. The core orchestrates failure attribution, signal–policy search, counterfactual evaluation, and sequential portfolio construction. The adapter exposes the host agent's runtime boundaries, observable information, feasible actions, and executable intervention semantics. This separation allows the same optimization procedure to operate within different agent environments while constraining proposals to feasible local controllers.

Structured failure attribution and proposal. An optimization model supports autonomous learning in two roles. As a failure attributor, it analyzes residual trajectories and identifies consequential runtime decisions, including earlier decisions implicated in downstream failures. As a controller proposer, it constructs candidate anchors $A = (\ell, \phi, \mu)$ within the feasible spaces $\mathcal{C}_\ell(H, \Phi)$ at the selected boundaries $\ell \in \mathcal{L}_c$. The same model may perform both roles.

Unlike open-ended harness-edit proposals, each candidate must specify where control is exercised, which locally observable conditions determine its activation, and what executable intervention follows. The optimization model proposes these conditional controllers, but the runtime evaluates their conditions and enforces their control flow. Their downstream value is established through empirical evaluation, not the optimization model's assessment.

Host model and deployment. The host model is the fixed language model whose agent behavior is being optimized; it may be identical to or different from the optimization model. After learning, the deployed system consists of the host model, runtime adapter, and accepted controller portfolio. The optimization core and optimization model are not required during ordinary inference unless explicitly invoked by a deployed controller.

4 Experiments

We first evaluate end-to-end task performance on BFCL Agent Memory (Patil et al., 2025) and AppWorld (Trivedi et al., 2024), using BFCL as a running example to examine learned controllers and

Model	Split	H_0	GEPA	Self-Harness	AnchorOpt
MiniMax M2.5	train	23.7%	34.0% (+10.3 pp)	H_0 retained	26.0% (+2.2 pp)
	held-out	42.7%	58.7% (+16.0 pp)	H_0 retained	46.7% (+4.0 pp)
Qwen3.6-35B	train	39.8%	39.3% (−0.5 pp)	H_0 retained	H_0 retained
	held-out	19.1%	35.7% (+16.7 pp)	H_0 retained	H_0 retained
Opus → Granite 4.1-8B	train	30.0%	31.0% (+1.0 pp)	31.9% (+1.9 pp)	37.6% (+7.6 pp)
	held-out	16.7%	19.1% (+2.4 pp)	19.8% (+3.1 pp)	21.4% (+4.8 pp)

Table 2: BFCL v4 Agent Memory strict task success (%). Full provenance is in Appendix B.1.

Model	Split	H_0	GEPA	Self-Harness	AnchorOpt
MiniMax M2.5	train	65.6%	71.1% (+5.5 pp)	68.9% (+3.3 pp)	70.6% (+5.0 pp)
	held-out	54.4% / 55.2%	62.8% (+8.3 pp)	61.7% (+7.2 pp)	60.6% (+5.4 pp)
Qwen3.6-35B	train	75.6%	71.7% (−3.9 pp)	H_0 retained	H_0 retained
	held-out	58.9%	51.1% (−7.8 pp)	H_0 retained	H_0 retained
Granite 4.1-30B	train	30.0%	H_0 returned	H_0 retained	33.3% (+3.3 pp)
	held-out	26.7%	H_0 returned	H_0 retained	30.0% (+3.3 pp)

Table 3: AppWorld task-goal completion (TGC, %). Further details appear in Appendix D.1.

their relationship to global prompt and harness optimization. We then investigate repeated-execution reliability on τ-bench (Yao et al., 2024), and constraint compliance and intervention cost on CCTU (Ye et al., 2026b). Finally, we examine autonomous versus human-guided controller discovery.

Experimental setup. We compare the native harness H_0, GEPA (Agrawal et al., 2025) (global prompt optimization), Self-Harness (Zhang et al., 2026) (open-ended harness edits), and AnchorOpt (structured local controllers).

We study MiniMax M2.5 (Chen et al., 2026), Qwen3.6-35B-A3B (Qwen Team, 2026), and IBM Granite 4.1-8B and 4.1-30B (IBM Research, 2026). The host model executes the agent, while the optimization model diagnoses failures and proposes modifications. Unless otherwise specified, the same model serves both roles; "Opus → Granite 4.1-8B" denotes Claude Opus 5 (Anthropic, 2026) as the optimization model and Granite as the host.

4.1 End-to-End Task Performance on BFCL and AppWorld

Tables 2 and 3 compare AnchorOpt with the native harness H_0, GEPA, and Self-Harness on BFCL Agent Memory and AppWorld. We report task success and percentage-point changes relative to each method's own native reference, emphasizing frozen held-out evaluation. "H_0 retained" means that no candidate modification was deployed, while "H_0 returned" means that GEPA selected its original prompt unchanged. Neither is a measured zero treatment effect. Experimental configurations and per-arm provenance are detailed in Appendix B.

Comparison with Self-Harness: Why structured runtime optimization matters. Self-Harness is our closest conceptual comparison: both methods optimize the execution harness around a fixed language model. However, Self-Harness searches open-ended source-level modifications, whereas AnchorOpt restricts search to boundary-conditioned controllers with explicitly observable signals and feasible actions.

The results illustrate the value of this structured search space. On BFCL with MiniMax, Self-Harness retains the native harness, while AnchorOpt improves held-out strict success by 4.0 percentage points. On AppWorld with Granite 4.1-30B, Self-Harness again retains H_0, while a learned AnchorOpt controller improves held-out TGC by 3.3 points.

By restricting proposals to consequential runtime boundaries and executable local actions, AnchorOpt avoids searching over arbitrary source edits and can evaluate controller activation and downstream value against the current incumbent. The AppWorld controllers illustrate this mechanism: both act after generation but before tool execution, addressing proposed actions that could otherwise

Local controller	Targeted failure	Result on top of GEPA
Vector read-side	Insufficient retrieval	No net gain: $29/104 \rightarrow 29/104$
Key–value capacity	Rejected memory writes	$+9$ successes on the targeted backend
Vector write-side	Relevant facts never stored	$29/104 \rightarrow 47/104$ and $25/104 \rightarrow 48/104$

Table 4: Interaction between local runtime control and a frozen GEPA-optimized prompt on MiniMax BFCL. The read-side comparison is store-fixed; the write-side results come from two separate fresh-build comparisons. The write-side controller is exploratory and was not included in the final accepted portfolio. Results are not frozen held-out portfolio gains.

produce incomplete data or target nonexistent APIs. The detailed intervention comparisons appear in Appendix D.1.

comparison with GEPA: Why local control matters. GEPA provides a different and complementary optimization surface: it learns a shared prompt that can substantially improve overall agent behavior. On BFCL with Qwen, GEPA raises held-out success from 19.1% to 35.7%. However, this aggregate improvement conceals heterogeneous effects across memory backends. Recursive-summary improves from $7/20$ to $17/20$ and vector memory from $6/40$ to $11/40$, while key–value memory falls from $3/24$ to $2/24$. On the training corpus, opposing backend changes of -13, $+10$, and $+2$ cases nearly cancel in the aggregate.

These results motivate finer-grained control rather than showing that global prompt optimization is ineffective. GEPA's learned prompt can itself contain backend-specific instructions, but the shared prompt is applied across queries and its execution is mediated by the host model. In contrast, an AnchorOpt controller specifies a runtime boundary and an explicit activation condition, allowing an intervention to be dispatched only in the local states for which it was designed. Other states follow native execution unless affected indirectly through subsequent trajectory or state changes. Thus, local control offers a way to target heterogeneous residual failures without requiring the same intervention to be applied throughout the agent's execution.

When do global and local optimization complement each other? GEPA and AnchorOpt optimize different surfaces and can be combined sequentially. We freeze the GEPA-optimized prompt, re-mine its residual failures, and evaluate local controllers against the resulting incumbent:

$$H_0 \longrightarrow H_{\mathrm{GEPA}} \longrightarrow H_{\mathrm{GEPA+AnchorOpt}}.$$

The MiniMax BFCL experiments reveal that the marginal value of local control depends on whether it addresses an overlapping or distinct failure mechanism (Table 4).

Overlapping mechanisms. A vector read-side controller improves the native host but adds no net success on top of GEPA in a matched store-fixed comparison ($29/104$ for both arms). GEPA's learned prompt already encourages retrieval verification, consistent with a reduced marginal benefit from another read-side intervention.

Distinct residual bottlenecks. A key–value capacity controller instead addresses rejected memory writes and improves success by nine cases on its targeted backend against a frozen GEPA incumbent. A separate exploratory vector write-side controller provides a clearer example of complementarity: it improves whether relevant facts are stored, while GEPA's instructions support their subsequent retrieval and use. The write-side intervention alone increases fact availability without a corresponding task-performance gain. Combined with GEPA, however, it yields 18 and 23 additional successes in two fresh-build comparisons.

These results illustrate a mechanism-based principle for combining optimization methods: local controllers can add value when they address residual bottlenecks outside the behavior improved by a global prompt, but may provide little additional benefit when the two interventions target the same stage of execution. The write-side findings are exploratory rather than accepted-portfolio or frozen held-out effects; controller-level attribution and measurement qualifications appear in Appendix C.4.

Metric	H_0	AnchorOpt	Gain (pp)
pass^1	0.188	**0.350**	+16.2
pass^4	0.000	**0.350**	+35.0

Table 5: Repeated-execution reliability on τ-bench with Granite 4.1-30B on airline tasks ($n = 20$). pass^k measures success in all k executions of the same task.

4.2 Repeated-Execution Reliability on τ-bench

End-to-end task success does not reveal whether an agent succeeds consistently across repeated executions. On τ-bench (Yao et al., 2024), we measure pass^k, the fraction of tasks successful in all k repeated trials.

Table 5 illustrates the reliability benefit of learned runtime control. On Granite 4.1-30B airline tasks, AnchorOpt raises single-run success by 16.2 percentage points and four-run reliability by 35.0 points. Seven of twenty tasks succeed in all four trials under AnchorOpt, compared with none under the native harness. Across nine matched host–domain cells, AnchorOpt improves mean single-run success by 3.6 percentage points. Full per-cell results and comparisons with GEPA and Self-Harness appear in Appendix D.2.

4.3 Constraint Mitigation and Intervention Cost on CCTU

Task success alone does not capture whether an agent respects constraints along the way. In consequential workflows, an invalid action may incur costs or irreversible effects even if the agent eventually completes the task. CCTU (Ye et al., 2026b) provides a setting to examine whether runtime control can mitigate violations before a proposed response is committed.

We conduct a focused ablation with one state-conditioned controller on Qwen3.6-35B. At the post-generation/pre-execution boundary, the controller detects a drafted response that exceeds its length limit and selects an intervention based on the agent's tool-use state. This local intervention reduces total constraint violations by 11.6% on training episodes and 8.8% on held-out episodes, including reductions in some untargeted constraint classes.

Why intervention frequency matters. More frequent intervention does not necessarily produce a better runtime policy. On the training set, limiting each controller branch to one activation per episode removes 188 response-length violations while introducing 23 redundant identical tool calls. Intervening on every eligible turn removes 326 violations but introduces 409 redundant calls. Thus, the additional violation reduction comes with substantially greater execution overhead.

The experiment illustrates an important feature of decision-local optimization: intervention policies must be evaluated by their downstream consequences, not merely by the number of local violations they prevent. Task solve rate and perfect solve rate remain unchanged in this ablation, so the results demonstrate selective constraint mitigation rather than complete constraint enforcement or improved end-to-end task success. They motivate extending the optimization objective to account jointly for task reward, constraint violations, and intervention cost. The full results and controller specification appear in Appendix E.

4.4 Autonomous Discovery and Human Guidance

We investigate whether AnchorOpt can autonomously identify consequential runtime decisions and construct effective local controllers. Our BFCL study includes a human-guided progression, in which an author proposes conditions and interventions from residual trajectories, and a blind autonomous progression, in which the learner receives no prior controller identities, signals, policies, or reported gains. Both progressions yield improvements, with the human-guided progression achieving a larger training gain. Their distinct evaluation results are reported in Appendix F.

Different failure mechanisms discovered. Human-guided search identified a broad range of failures, including capacity limitations, missing or duplicate identifiers, destructive operations, and weak

retrieval similarity. Many of these conditions correspond to explicit errors or recognizable execution patterns. The autonomous learner independently discovered a thin-retrieval-content condition, which arises when a tool call succeeds but returns insufficient evidence for the downstream task. It learned to invoke an existing cross-container merge selectively, addressing a failure that produces no explicit tool error.

The autonomous learner also rediscovered an archived no-tool-call controller without being given its original condition or intervention. These results illustrate partly overlapping but distinct discovery paths: human-guided search explored a broader collection of failure mechanisms, while autonomous search independently recovered useful controller structures and identified a condition based on the quality of a successful operation. The thin-content condition is related to the human-guided weak-similarity condition, rather than an entirely unprecedented failure category.

Human–AI collaboration and deployment. At the present stage, the two approaches are complementary. An optimization model can repeatedly analyze residual trajectories, propose local controllers, and evaluate alternatives against the evolving incumbent. Human guidance remains valuable for interpreting unexpected results, identifying missing adapter capabilities, and distinguishing ineffective interventions from controllers that never executed. Our audits uncovered boundary mismatches, unavailable signals, and unsupported actions that could otherwise appear as unsuccessful policy proposals.

Crucially, this collaboration need not increase the cost of every subsequent agent execution. In our heterogeneous BFCL setting, Claude Opus supports failure attribution and controller proposal for Granite 4.1-8B. Once the controllers are learned and installed, the optimization model is no longer required during ordinary inference: the runtime evaluates their conditions and dispatches interventions around the fixed Granite host.

The potentially expensive discovery process is therefore performed offline, while the resulting controller portfolio can be reused across subsequent tasks. Depending on the application's execution volume, cost structure, and stability of its failure distribution, this separation may make frontier-model-assisted optimization attractive even when deploying the frontier model for every agent interaction would be impractical.

5 Discussion and Conclusion

AnchorOpt offers a structured approach to agent harness optimization by treating consequential runtime decisions as the units of optimization and deployment. Boundary-conditioned feasibility, alternating signal–policy search, and residual-driven sequential portfolio construction decompose open-ended harness design into smaller, executable decision problems evaluated by downstream task performance.

Across agent benchmarks, learned controllers improve task success and, in a repeated-execution study, reliability. A focused constraint-mitigation ablation further illustrates the need to account for intervention cost alongside local benefit. Human-guided and autonomous discovery identify partly distinct failure mechanisms, suggesting complementary roles in developing effective runtime controllers.

The structured controller space also limits expressiveness, and sequential optimization may miss beneficial joint modifications. Empirical acceptance does not guarantee safety or generalization; adaptive selection requires an untouched final evaluation. Some cross-method BFCL results use different native configurations, and autonomous discovery remains constrained by the adapter's available signals and actions.

An important practical consequence is the separation of optimization from deployment: a stronger model can assist controller discovery offline, while the accepted portfolio is reused around a smaller fixed host without requiring the optimization model at inference time. This positions structured runtime optimization as a reusable interface for improving agent behavior beyond model weights and global instructions.

AI USE STATEMENT

Generative AI tools, primarily Claude, were used extensively to support software development and manuscript preparation. Claude assisted with implementing and refining harness adapters, runtime controllers, evaluation infrastructure, telemetry, analysis scripts, and experiment orchestration. This was an iterative, author-directed development process: the authors specified the research objectives, mathematical formulation, algorithmic design, runtime interfaces, and experimental protocols; reviewed and corrected generated implementations; and verified experimental behavior and reported results against execution traces and evaluation artifacts. AI tools also assisted with summarizing experimental records and organizing and editing manuscript text.

The research contributions, experimental decisions, interpretations, and conclusions are the authors' responsibility. Generative AI models also serve as experimental components in this work, including as optimization-time failure attributors and controller proposers and as host models executing agent tasks. These uses are part of the proposed method and its evaluation, distinct from AI assistance in developing the research software and manuscript.

ETHICS STATEMENT

AnchorOpt modifies the execution behavior of tool-using agents, including decisions that can affect persistent state. Incorrect interventions may introduce new failures or alter downstream behavior. We therefore evaluate candidate controllers against the current incumbent and examine both improvements and regressions. These empirical checks do not constitute formal safety guarantees. Deployment in consequential settings would require appropriate permissions, validation, and human oversight.

REPRODUCIBILITY STATEMENT

Section 3 describes the method, with implementation and evaluation details provided in Appendices A and B. Our anonymized code release includes the optimization core, runtime adapters, and experiment scripts: `https://anonymous.4open.science/r/anchoropt-7644`.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Anthropic. Introducing claude opus 5. Anthropic, July 2026. URL `https://www.anthropic.com/news/claude-opus-5`.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, et al. The minimax-m2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026.

Jizhou Chen and Samuel Lee Cong. Agentguard: Repurposing agentic orchestrator for safety evaluation of tool orchestration. *arXiv preprint arXiv:2502.09809*, 2025.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Nicolas Heess, Mohammad G. Azar, Bilal Piot, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight credit assignment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

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

IBM Research. Introducing the IBM Granite 4.1 family of models. IBM Research, April 2026. URL https://research.ibm.com/blog/granite-4-1-ai-foundation-models.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, and Sami Jaghouar. Prime agent: A self-improving RLM harness. *arXiv preprint arXiv:2608.23552*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

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.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, 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.

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.

Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, Éric Moulines, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2021.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026.

Theodore Papamarkou, Pierre Alquier, Matthias Bauer, Wray Buntine, Andrew Davison, Gintare Karolina Dziugaite, Maurizio Filippone, Andrew YK Foong, Vincent Fortuin, Dimitris Fouskakis, et al. Position: agentic ai orchestration should be bayes-consistent. *arXiv preprint arXiv:2605.00742*, 2026.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, et al. Autosaddler: Automatic harness optimization with durable updates from agent execution traces. *arXiv preprint arXiv:2608.23041*, 2026.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations (ICLR)*, 2024.

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.

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 (NeurIPS)*, 2023.

Wei Sun. Decision-centric design for LLM systems. *arXiv preprint arXiv:2604.00414*, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Asterios Tsiourvas, Wei Sun, and Georgia Perakis. Causal LLM routing: End-to-end regret minimization from observational data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. *arXiv preprint arXiv:2503.18666*, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. *arXiv preprint arXiv:2601.21557*, 2026a.

Junjie Ye, Guoqiang Zhang, Wenjie Fu, Tao Gui, Qi Zhang, and Xuanjing Huang. Cctu: A benchmark for tool use under complex constraints. *arXiv preprint arXiv:2603.15309*, 2026b.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, 2025.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

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*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025b.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024.

A IMPLEMENTATION OF ANCHOROPT

This appendix specifies how the boundary-conditioned controller space in Section 3 is implemented. The current harness is $H = H_0 \oplus \mathcal{P}$, and a candidate controller is $A = (\ell, \phi, \mu)$. The pseudocode below describes the general optimization procedure; the experimental sections distinguish complete portfolios from focused single-controller studies.

A.1 ADAPTER CONTRACT AND EXECUTED INTERVENTIONS

The environment adapter exposes the boundary types $\mathcal{L}(H)$, locally available observations $\mathcal{O}_\ell^H$, and executable actions $\mathcal{U}_\ell^H(o)$. A boundary type can recur at multiple points in an execution. A controller may use current context, persistent state, a proposed call, or a realized tool result only when that information exists at its declared boundary; in particular, an upstream controller cannot read a future tool error. Signals may combine structured metadata, text, history, and estimated quantities, provided their inputs are locally observable.

The adapter implements five operator classes. NOOP continues native execution; REPROMPT requests a new host-model response with feedback; SUPPRESS prevents a pending call; SUBSTITUTE replaces a pending call or its arguments; and TRANSFORM modifies admissible context, observations, or persistent state passed downstream. A completed call cannot be undone by later suppression or substitution. The adapter must check the availability of the requested action at the declared boundary and fall back to NOOP when no permitted intervention applies. Installed controllers execute in portfolio order, with adapter-defined conflict resolution when more than one applies.

The runtime, rather than the host model, evaluates the condition and dispatches the action. This guarantees that an executable intervention is issued when its condition holds; it does *not* guarantee that the host complies with a natural-language REPROMPT. The optimization model can propose a controller offline without participating in ordinary execution of the accepted portfolio.

A.2 CANDIDATE EVALUATION AND ACCEPTANCE

A failure attributor identifies candidate consequential boundaries from residual trajectories under the *current* incumbent. Its hypotheses focus search; they do not establish that a repair works. Proposal, executability, actual non-NOOP intervention, and downstream success are separate observations. For each evaluated candidate, we record its declared boundary, signal, and action; whether the condition is eligible; whether the action executes; and gains and regressions against the current portfolio.

The promotion quantity is the whole-corpus marginal value $\widehat{\Delta}_d(A \mid \mathcal{P}_k)$ from Eq. (4), computed against the incumbent on the same evaluation tasks. Trigger-filtered outcomes are useful for diagnosing the mechanism, but are not a substitute for whole-corpus evaluation: changed state can affect later episodes or decisions without an additional firing at the affected point. Performance, execution validity, and prespecified resource checks determine promotion; a candidate that merely fires is not thereby accepted. Data used to choose or promote a controller are not an untouched final held-out evaluation.

If an intervention begins after a reusable trajectory prefix, an evaluator may cache that prefix only if the candidate cannot affect it and the complete resumption state can be restored. Otherwise, the candidate is executed under its own harness. An upstream persistent-state change can alter the entire continuation, including opportunities for later controllers to fire.

A.3 FULL OPTIMIZATION PROCEDURE

Let $\mathcal{T}$ be trajectories under the incumbent, $\mathcal{F}$ its diagnosed residual failures with supporting evidence, and $\mathcal{L}_c \subseteq \mathcal{L}(H)$ the consequential boundaries localized from those failures. The policy search proposes a set $\mathcal{A}_c$ of feasible controllers from $\mathcal{C}_\ell(H, \Phi)$ at these boundaries. EVALUATEANDSELECT returns an accepted controller A^* or $\varnothing$ when no evaluated candidate meets the promotion criteria. The budget B bounds optimization effort; the reported experimental arms use different budgets and portfolio sizes as identified in their protocols.

Algorithm 1 AnchorOpt: residual-driven optimization

Require: Native harness H_0, initial signals Φ_0, budget B

$\mathcal{P} \leftarrow \emptyset$, $\Phi \leftarrow \Phi_0$
$\mathcal{T} \leftarrow \textsc{Execute}(H_0)$
while budget remains **do**
  $H \leftarrow H_0 \oplus \mathcal{P}$
  $\mathcal{F} \leftarrow \textsc{DiagnoseResiduals}(\mathcal{T})$
  if $\textsc{Stop}(\mathcal{F})$ **then**
    return $\mathcal{P}$
  $\mathcal{L}_c \leftarrow \textsc{ConsequentialBoundaries}(\mathcal{F}, H)$
  $\mathcal{A}_c \leftarrow \textsc{SearchFeasiblePolicies}(\mathcal{L}_c, \Phi, H)$ ▷ Fix Φ
  $A^* \leftarrow \textsc{EvaluateAndSelect}(\mathcal{A}_c, H)$
  if $A^* \neq \varnothing$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \oplus A^*$
    $\mathcal{T} \leftarrow \textsc{Execute}(H_0 \oplus \mathcal{P})$
    continue ▷ Re-mine residuals
  if budget exhausted **then**
    return $\mathcal{P}$
  $\Phi' \leftarrow \textsc{ExpandSignals}(\mathcal{F}, \mathcal{L}_c, \Phi)$ ▷ Fix $\mathcal{P}$
  if $\Phi' = \Phi$ **then**
    return $\mathcal{P}$
  $\Phi \leftarrow \Phi'$
return $\mathcal{P}$

Search under a fixed signal representation can accept multiple controllers before representation expansion. After each acceptance, the portfolio changes, trajectories are regenerated, and residual failures are re-mined. Changing the runtime boundary and enriching a signal at the same boundary are distinct operations: the former changes observable information and feasible actions, whereas the latter changes the representations searched within that space.

A.4 Richer Signals, Policies, and Objectives

The interface is not restricted to one Boolean rule per failure type. For example, the condition `backend=rec_sum` $\wedge$ `error_kind=blob_would_overflow` is a conjunction within one signal representation. Contextual and textual estimates could also be combined, as in the decision-centric formulation of Sun (Sun, 2026):

$$\widehat{s}_\alpha(o) = \alpha s_{\text{context}}(o) + (1-\alpha) s_{\text{text}}(o), \qquad \alpha \in [0,1]. \tag{5}$$

Given such a signal, the policy could choose an available recovery action only when `backend=vector` and $\widehat{s}_\alpha(o) < \theta$, and choose NoOp otherwise. Signal construction and policy-parameter search are separate optimization operations. This equation illustrates an admissible extension, not an additional accepted controller in the reported BFCL results.

The reported task-performance experiments primarily optimize success. Constraint violations, intervention frequency, latency, or tool/model cost can instead enter the objective or acceptance constraints. The focused CCTU study in Appendix E illustrates why minimizing a local violation count alone is insufficient to establish improved task utility.

B Experimental Setup and Baseline Protocols

B.1 Benchmarks, Measures, and Splits

BFCL v4 Agent Memory. We use strict task success: an episode must satisfy the benchmark's official validity requirements without an exception. The three memory backends are key–value, recursive-summary, and vector. The principal study arms have different, explicitly recorded training and frozen held-out sets: MiniMax uses 312 training and 75 held-out queries, Qwen uses 211 and 84, and Opus $\rightarrow$ Granite 4.1-8B uses 303 and 84 for the reported AnchorOpt portfolio. The MiniMax

held-out queries come from a withheld healthcare scenario, with 25 queries per backend. The Granite training portfolio in Table 2 is measured on 303 cases; other Granite method arms may use a 211-case comparison subset. Consequently, neither a training percentage nor an independent native reference should be transferred from one arm to another.

The held-out Granite AnchorOpt result in the main table is $16.7\% \rightarrow 21.4\%$; its training result is $30.0\% \rightarrow 37.6\%$. The MiniMax portfolio is $23.7\% \rightarrow 26.0\%$ on training and $42.7\% \rightarrow 46.7\%$ on held-out. These are the portfolio outcomes, not a sum of separate candidate-screening gains. The Qwen AnchorOpt search retained H_0; an H_0-retained cell is not an estimated zero treatment effect.

AppWorld. Each evaluation pass contains 90 tasks arranged in 30 scenarios. We report task-goal completion (TGC): the fraction of individual tasks whose goals pass the benchmark's programmatic checks. Scenario-goal completion (SGC) requires all tasks in a scenario to succeed and is therefore a different, less granular estimand. The present results do not establish an SGC improvement; the main table reports TGC throughout.

The MiniMax AnchorOpt held-out score is $109/180 = 60.6\%$ across two passes, compared with its $149/270 = 55.2\%$ native reference across three. The GEPA and Self-Harness MiniMax comparison in the main table uses a separately pooled 54.4% native reference. Granite's reported AnchorOpt held-out comparison is $48/180 \rightarrow 54/180$ ($26.7\% \rightarrow 30.0\%$). These are measured point estimates based on the recorded passes, not claims of identical repetition protocols across all methods. The Qwen AnchorOpt held-out treatment is *not measured*; its available native and treatment records lack the same completion-cap configuration and cannot supply a valid matched comparison.

Other studies. The τ-bench experiment uses the official airline, retail, and telecom train/test splits (respectively $30/20$, $74/40$, and $74/40$ tasks), with repeated executions to compute pass^k. The CCTU ablation uses Qwen3.6-35B, 140 training and 60 held-out tasks, each evaluated twice, with a single state-conditioned controller. CCTU violations are counted from validator outputs; task accuracy, solve rate (SR), and perfect solve rate (PSR) are separate outcomes.

B.2 What the Baselines Change and How Their Results Are Read

GEPA optimizes a shared instruction/preamble surface, selecting a prompt using its internal validation criterion. A selected prompt can contain backend-specific instructions, but the preamble is injected globally and its effect is mediated by host-model behavior. "H_0 returned" denotes selection of the original prompt. Self-Harness searches a broader space of source-level harness edits, potentially including conditional behavior, and uses its own proposal and promotion pathway. "H_0 retained" denotes that no candidate change was deployed; it need not mean that a zero-effect candidate was measured. AnchorOpt proposes explicitly located, executable local controllers and evaluates their marginal value against its evolving incumbent.

The selection rules also affect how outcomes are read. GEPA deploys the prompt chosen on its internal validation criterion; a negative change on a subsequent held-out evaluation is not automatically screened out. By contrast, Self-Harness and AnchorOpt may retain the incumbent after a candidate fails their respective promotion checks. Thus, "H_0 retained" can mean either that no deployable candidate was found or that an evaluated candidate was rejected; it is not a measured zero treatment effect or a direct measure of proposal quality. Selection surfaces and promotion criteria differ across the reported campaigns and are not interchangeable.

Tables 2 and 3 summarize the reported outcomes but should not be interpreted as fully paired, identically initialized four-way experiments in every row. In particular, Granite BFCL methods have separately recorded native harness configurations and training surfaces; comparing their absolute held-out percentages does not isolate a method-only treatment effect. MiniMax AppWorld native values also differ with the number of pooled baseline passes. We retain the main paper's results and present method-specific changes relative to each recorded native reference, rather than importing figures from older, separate campaigns.

BFCL backend	Native	GEPA	Net cases
Key–value	$26/73$	$13/73$	-13
Recursive-summary	$41/76$	$51/76$	$+10$
Vector	$17/62$	$19/62$	$+2$

Table 6: Qwen BFCL training results under the shared GEPA preamble. The near-zero aggregate change conceals backend-specific gains and losses; this table is not the frozen held-out comparison.

Example	Local condition	Boundary	Intervention
Vector no-tool-call	Vector query without a tool call	Query / pre-execution	Targeted REPROMPT
Key–value capacity	Rejected or capacity-blocked write, with available recovery state	Compatible capacity-recovery gate	Re-address or recover the write
Granite thin-content	Successful vector retrieval returns insufficient usable content	Post-execution	Selective cross-container merge

Table 7: Illustrative BFCL conditional controllers. Exact signal implementations and executable actions depend on the adapter and host.

B.3 Evaluation and Measurement Qualifications

Controllers are promoted on their specified selection surfaces, followed by frozen held-out evaluation where available. A development slice used for acceptance is not an independent test, and rejected or unexecuted candidates do not establish a measured null. On AppWorld and τ-bench, identical configurations can vary across executions even when the model temperature is zero; the number of baseline and treatment passes therefore matters to the precision of small differences. In interpreting a local controller, we separate its firing and actual action execution from the whole-task result, and do not use a trigger-filtered gain in place of held-out task success.

C BFCL: Runtime Mechanisms and Complementarity

C.1 Backend Heterogeneity Under a Shared Prompt

The Qwen GEPA experiment makes the distinction between aggregate and backend-level improvement concrete. Its selected preamble identifies the backend, recommends backend-specific retrieval, and requests verification of tool results; it is not a backend-agnostic instruction. Nonetheless, it is applied as one shared prompt rather than dispatched at selected runtime boundaries. On the 211-case training evaluation, aggregate success is nearly unchanged ($84 \rightarrow 83$), while the backend changes have different signs:

On the frozen held-out split, the corresponding backend outcomes are key–value $3/24 \rightarrow 2/24$, recursive-summary $7/20 \rightarrow 17/20$, and vector $6/40 \rightarrow 11/40$. The overall held-out result, $19.1\% \rightarrow 35.7\%$, is reported in Table 2. These observations motivate state-specific intervention rather than implying that global prompt optimization is ineffective or that backend-specific instructions cannot be written into a global prompt.

C.2 Examples of Located, Conditional Control

Table 7 illustrates the location–signal–action structure. Conditions must use information visible at the specified boundary; the action must be executable there. The examples describe observed controller structures rather than a claim that the same policy transfers unchanged across all hosts and memory backends.

The activation condition is part of the learned decision. In the MiniMax BFCL controller-screening comparisons, the same read-side recovery instruction was accepted with its vector-memory condition (A1) but rejected under a key–value condition (K2). Conversely, the capacity-recovery instruction was accepted for key–value memory (K1) but rejected with a vector-memory condition

(V3). These comparisons hold intervention text fixed while changing the targeted backend; they are screening outcomes on different eligible populations, not an independent held-out experiment that isolates the condition's causal effect. They illustrate why a local intervention cannot be validated by optimizing its text without its activation condition.

C.3 Backward Localization and Signal Expansion in BFCL

Locating the consequential decision. An incorrect final answer can arise from different earlier decisions, which call for interventions at different boundaries. For example, a fact missing at retrieval may never have entered memory because an earlier key–value write was rejected at capacity. Recovery must then address the write or its available capacity-recovery action, using information observable at that boundary; changing only the later retrieval response cannot create the missing stored fact. In the Granite thin-content example in Table 7, the retrieval itself succeeds, but its returned content is insufficient for answering. The learner instead identifies a post-execution decision where an existing cross-container merge can be dispatched before the answer. In its accepted round, this thin-content controller adds three successful training tasks and two successful vector-development tasks, without observed paired losses; execution records confirm additional merges in treatment. The development slice participated in selection and is not a separate frozen test. Unlike an error-code trigger, this condition evaluates the adequacy of a successful retrieval. These examples illustrate why failure visibility does not by itself specify the intervention boundary. The proposed boundary and action still require execution and downstream evaluation; an earlier capacity-recovery prototype triggered but requested an action unavailable at its declared boundary, so its apparent null was not a test of an executed repair.

Expanding signals after policy search stalls. The human-guided progression in Table 11 records an explicit representation-expansion step. After capacity, missing-identifier, and duplicate-identifier interventions, the incumbent reaches 38.61% training and 22.62% development success at T3. At T4, no policy over the then-available signals passes the acceptance checks; the controller portfolio and both scores remain unchanged while the signal representation is expanded. Subsequent rounds include no-tool-call recovery (T5: 42.24% train, 26.19% development) and eventually a weak-retrieval condition (T9: 52.48% train, 48.81% development). These are sequential incumbent scores, not isolated causal effects of signal expansion. T4 belongs to the human-guided progression; the separate blind autonomous results do not establish that the learner independently performed this same expansion step.

C.4 MiniMax Controllers on a Frozen GEPA Incumbent

To test complementarity, we freeze GEPA's selected prompt, inspect its remaining failures, and evaluate candidate local interventions against that incumbent. The experiments distinguish interventions that overlap with GEPA's behavior from interventions that change another stage of the memory workflow.

A read-side vector controller helps the unoptimized host in a store-fixed comparison, but adding it to GEPA yields no additional vector success in the matched comparison ($29/104$ with or without the controller, with verified firings). This is consistent with overlap between the controller's retrieval-oriented action and GEPA's retrieval-verification instructions. In contrast, a key–value capacity intervention improves the backend where it executes by nine cases against a frozen GEPA incumbent.

A separate exploratory *write-side* vector intervention addresses whether relevant facts are stored, rather than whether retrieved evidence is used. In two fresh-build comparisons with a fixed GEPA program, the combined configuration moves $29/104 \rightarrow 47/104$ and $25/104 \rightarrow 48/104$ relative to its corresponding controls ($+18$ and $+23$ cases). An accompanying store audit records increased availability of needed facts ($34 \rightarrow 55$ cases), consistent with the write-side intervention changing what can subsequently be retrieved. In a separate unoptimized-host comparison, more available facts did not translate into additional task success; storage and use of evidence are therefore distinct potential bottlenecks. This controller was not part of the final accepted portfolio; the comparisons are not an untouched held-out evaluation of an optimized GEPA-plus-AnchorOpt stack. They illustrate a plausible complementary mechanism rather than establishing a general additive treatment effect.

Host	Targeted proposed action	Intervention comparison
MiniMax M2.5	Proposed step at risk of proceeding with incomplete data	The selected state-conditioned intervention passed its recorded repeated-baseline promotion rule; generic same-boundary and suppression alternatives did not.
Granite 4.1-30B	Proposed call to an unknown API path	Several response variants were screened at the same boundary and condition; the selected variant and a bare-suppression alternative both changed task outcomes.

Table 8: AppWorld pre-execution interventions. These are mechanistic screening contrasts, not additional frozen held-out portfolio estimates.

Individual candidate gains also need not add under sequential composition: controllers can change persistent state and the downstream opportunities for other controllers to act.

D Additional AppWorld and Reliability Results

D.1 AppWorld Controller Examples and Variability

The accepted MiniMax and Granite controllers both intervene after model generation and before execution of a proposed action. This boundary allows them to inspect a proposed operation and select an executable alternative before the agent commits to it. Table 8 summarizes the local interventions; screening comparisons concern individual candidates and must not be added to the portfolio-level results.

The selected Granite response variant happened to contain language about pagination, although its condition identified an unknown API path. Its outcome therefore does not establish that pagination instructions repair unknown APIs: the measured treatment is the *complete* condition–boundary–intervention combination. In the held-in candidate screen, the selected Granite variant fired on 37 episodes and showed a $+6$ firing-attributed net but a $+3$ whole-corpus net; a no-text suppression alternative showed a $+5$ firing-attributed net. These are *screening diagnostics*, not additive effects or frozen held-out gains, and they do not isolate the effect of the pagination wording. Earlier post-execution retry variants did not survive the same campaign's screening.

The main AppWorld table reports the held-out MiniMax TGC change from its 55.2% native reference to 60.6%, and Granite from 26.7% to 30.0%. MiniMax and Granite have relatively few treatment and native repeats: nominally identical configurations can differ by roughly 2–5 successful tasks per 90-task pass. Consequently, small point-estimate margins—particularly Granite's $+3.3$ percentage points—remain sensitive to repeat sampling. The paper therefore makes a task-level TGC claim, not a claim of improved scenario-level completion or a uniformly successful intervention across all hosts. Qwen's AnchorOpt held-out result remains unmeasured under a comparable completion-cap protocol, as stated in Appendix B.

D.2 Repeated-Execution Reliability on τ-bench

For a task with c successes in n executions, pass^k is the fraction of k-trial subsets in which the task succeeds every time, $\binom{c}{k}/\binom{n}{k}$, averaged across tasks. The study covers nine host–domain cells and uses four trials per side for the reported AnchorOpt evaluations. Table 9 gives the corresponding single-run results. Each baseline framework is compared with its own native measurement; these separately sampled references should not be read as one shared paired control.

The Granite airline cell supports the focused reliability conclusion in Section 4.2: pass^1 increases from 0.188 to 0.350, while pass^4 increases from 0 to 0.350 (seven of twenty tasks succeed in all four runs). On the separate airline training split, the learned controller succeeds on $16/30$ tasks; five native-harness runs each succeed on 10–$12/30$. This replication supports the Granite-airline result but does not establish comparable reliability gains in other cells. Reliability is not uniformly improved: on MiniMax airline, pass^1 increases from 0.688 to 0.775 while pass^4 decreases from 0.600 to 0.450.

Host	Domain	H_0 (AnchorOpt arm)	AnchorOpt	GEPA Δ	Self-Harness Δ
MiniMax M2.5	airline	68.8	77.5	+3.8	+2.5
	retail	81.9	81.9	+1.2	−2.5
	telecom	98.1	98.1	0.0	+1.2
Qwen3.6-35B	airline	80.0	83.8	−5.0	−15.0
	retail	81.2	77.5	−8.8	−2.5
	telecom	99.4	100.0	0.0	0.0
Granite 4.1-30B	airline	18.8	35.0	+8.8	−2.5
	retail	69.4	73.1	−0.6	+4.4
	telecom	51.2	54.4	0.0	−2.5

Table 9: τ-bench held-out pass^1 (%). GEPA and Self-Harness columns show percentage-point changes relative to *their respective* native measurements, not the displayed AnchorOpt native column. The frameworks used different search depths and numbers of baseline repetitions; these cross-method deltas are contextual rather than a matched-budget comparison.

Measure	Training	Held-out
Total violations	$3726 \to 3294$ (-11.6%)	$1866 \to 1702$ (-8.8%)
Length violations	$786 \to 598$	$538 \to 488$
Accuracy (%)	$23.57 \to 22.86$	$20.00 \to 20.00$
Solve rate (%)	$9.29 \to 9.29$	$3.33 \to 3.33$
Perfect solve rate (%)	$6.43 \to 6.43$	$3.33 \to 3.33$

Table 10: CCTU: native harness → one state-conditioned controller. Violation counts aggregate two evaluations per task.

Thus, improvements in average task success do not by themselves establish repeated-execution reliability.

The nine-cell AnchorOpt mean gain in pass^1 is 3.6 percentage points, but the four-run reliability improvement is concentrated in the Granite airline setting. Two telecom cells are at or near ceiling, and some cell-level differences are comparable to variation across independent native runs. This study installs only one learned controller per cell, rather than a converged sequential portfolio; the baselines use their own optimization procedures and budgets. These features limit broader cross-method or cross-domain claims from the table.

E Constraint-Aware Runtime Control on CCTU

Dataset and controller. CCTU (Ye et al., 2026b) tests task completion under constraints on resource use, tool calls, arguments, and responses. We evaluate one human-authored, state-conditioned controller, not a complete autonomously learned constraint portfolio. On Qwen3.6-35B, the controller examines a drafted response at the post-generation/pre-execution boundary and checks whether it exceeds the task's response-length limit. If the agent has not yet called a tool, it asks the model to perform the necessary retrieval; after tool use, it instead emphasizes completing the task within the remaining interaction budget. The two branches are mutually exclusive and each fires at most once per episode in the deployed configuration.

Violation reduction and task outcomes. We use 140 training and 60 held-out tasks, with two evaluations per task. Table 10 separates violations from completion metrics. Total violations decrease on both splits, but solve rate and perfect solve rate are unchanged; training accuracy decreases slightly. The experiment therefore demonstrates selective mitigation of a local constraint failure, not comprehensive constraint enforcement or an increase in task completion.

Intervention-frequency ablation. On training episodes, the one-shot policy removes 188 length violations while introducing 23 redundant identical tool calls. Firing on every eligible turn removes 326 length violations but introduces 409 redundant calls. The additional violation reduction therefore incurs considerably more execution waste. This is a focused illustration of the need to evaluate

Round	Change	Training (%)	Dev. (%)
T0	Native harness	30.03	16.67
T1	Capacity recovery	33.66	19.05
T2	Missing-identifier recovery	34.65	19.05
T3	Duplicate-identifier suppression	38.61	22.62
T4	Expand signals; portfolio unchanged	38.61	22.62
T5	No-tool-call recovery	42.24	26.19
T6	No-remaining-slots recovery	43.56	34.52
T7	Blob-overflow handling	45.87	40.48
T8	Destructive-clear suppression	47.52	40.48
T9	Weak-retrieval response	52.48	48.81

Table 11: Human-guided BFCL progression on its development surfaces. These figures are not the blind autonomous portfolio's held-out results.

downstream task value and intervention cost jointly, rather than optimizing the number of local violations prevented.

The controller targets one observable condition and leaves other constraint classes unresolved. Its two evaluations per task use a deterministic single-worker setup; these repeated records should not be mistaken for a resampling-based estimate of generalization.

F Autonomous and Human-Guided Controller Discovery

F.1 Protocol and Distinct Roles

The human-guided BFCL progression uses author input to identify conditions and draft interventions from residual trajectories; the runtime still evaluates executable activation and promotes controllers against the current incumbent. The blind autonomous progression withholds earlier controller identities, conditions, policies, and measured gains from the learner. It may reuse matching frozen native trajectories, but must propose and evaluate new controllers through the live adapter. These are different learning conditions: the earlier human-guided progression is not folded into the blind autonomous portfolio, and its development results are not an independent held-out comparison with the autonomous arm.

An acceptance audit distinguishes a proposed controller, one that is feasible and installed, one that executes a non-NoOp action, and one that satisfies downstream acceptance criteria. Missing observations or actions at an adapter boundary can prevent a valid diagnosis from becoming an executable controller. Such cases are implementation or expressiveness limitations, not negative measurements of the intended intervention.

F.2 Human-Guided Progression

Table 11 retains the chronological *accepted-portfolio* record because it illustrates the sequential optimization and explicit signal-expansion step in Algorithm 1. Each row reports the resulting incumbent, not a gain measured for the named controller in isolation. T4 expands the available signal representation without changing the installed portfolio.

F.3 Blind Discovery Outcomes and Limits

The autonomous Opus $\rightarrow$ Granite result in the main BFCL table is a five-controller portfolio: training success increases from 30.0% to 37.6%, and held-out success from 16.7% to 21.4%. MiniMax's accepted two-controller portfolio increases held-out success from 42.7% to 46.7%; the Qwen workstream retains H_0. The human-guided progression yields the larger reported training gain, but its search conditions and evaluation surfaces are distinct, so this difference is not an estimate of the value of human guidance under a matched optimization budget.

The blind learner independently identifies a thin-content condition: a vector retrieval returns successfully but provides insufficient usable evidence, motivating a selective cross-container merge before

the final response. It also rediscovers an archived no-tool-call controller without being given its earlier condition or policy. The thin-content condition is related to the human-guided weak-retrieval signal, rather than a wholly unrelated failure category. These examples establish useful autonomous controller discovery while also showing why observed search failures must be separated from unavailable adapter signals, incompatible actions, and insufficient execution evidence. The optimization model is used for attribution and proposal offline; the resulting portfolio runs around the fixed host without requiring that model on every task.