From Traces to Guarded Programs: Evidence-Gated Compilation of Recurrent Agent Workflows

A Preprint

Reza Rahimi
JazzX AI
Los Altos, CA 94022, USA
reza.rahimi@jazzx.ai

August 9, 2026

Abstract

Tool-using agents repeat read-only evidence paths while paying for a model at every boundary. Deterministic programs can reduce cost, but recurrence is unsafe: arguments may lack provenance, tools may cross permissions, and a replay-correct prefix may still change the answer.

We introduce *guarded agentic compaction* (GAC), a trace-to-program compiler that reconstructs typed provenance, rejects unsafe effects, synthesizes from a closed 23-operator language, verifies empirical contracts, and applies exact finite-sample admission. Its normal output is refusal: compile only a supported prefix; otherwise retain the unchanged agent.

Across three GitHub workflows with live provider calls, compiled programs pass 90/90 exact contracts versus 89/90 for unchanged agents, with no compiled-only failure and a 3.3% pooled discordance upper bound. They reduce provider requests 50.0–75.0%, tokens 39.5–81.4%, latency 51.7–73.0%, and estimated cost 32.0–75.3%. Hand-written programs also reach 90/90 on the two new families, so the contribution is automatic discovery and calibrated admission, not runtime dominance.

A cross-repository extension seals five GitHub repositories, reaches 580/580 exact discovery traces, and completes 120/120 time-forward exact pairs on four while retiring the fifth. A balanced rerun over three repositories reaches 360/360 exact discovery traces and 180/180 exact held-out pairs, compacts `open`, `merged`, and `closed_unmerged` cases on all three, and reduces requests 66.7%, tokens 78.4%, observed wall latency 60.7%, and cost 72.7%. On NESTFUL and API-Bank, every recurrent family retires for insufficient support.

The central result is an evidence boundary: traces establish recurrence, not admissibility. Agent workflow optimizers should be evaluated by what they safely refuse as well as by what they accelerate.

Keywords LLM agents · tool use · workflow optimization · program synthesis · provenance · selective prediction · risk control · OpenAI Agents SDK

Code and artifacts. `https://github.com/rrahimi-uci/guarded-agentic-compaction`
Evidence. A ten-benchmark audit with two compiler substrates and six pilot-separated live-provider protocols on real public GitHub records.

1 Introduction

The standard reasoning–acting loop delegates every tool boundary to a language model: model, tool, model, tool, and so on [1]. This flexibility is valuable while a workflow is novel. At scale, however, mature agents often revisit the same

read-only evidence-gathering prefix with the same data dependencies. Repeating a model decision then consumes latency, tokens, and money without necessarily adding useful adaptation.

Deleting such decisions is not ordinary caching. A tool argument may depend on a prior observation, a repeated sequence may cross an approval or write, a nominal read may have a hidden external effect, and an input that resembles the training trace may violate a permission or freshness constraint. A fast wrong workflow is worse than a slow agent. The research problem is therefore:

> *Given historical executions of a tool-using agent, identify a deterministic region that can replace model-mediated control flow, and dispatch it only where observable evidence supports that replacement.*

This problem differs from current-query scheduling in LLMCompiler [2], prompt-program optimization in DSPy [3, 4], graph compression in AgentSlimming [5], model routing [6], and prompt compression [7]. It is closest to Agent Workflow Optimization (AWO), which converts frequent trace sequences into deterministic meta-tools [8], and to plan caching [9]. Our focus is the missing admission argument: typed value provenance, effect barriers, compatibility pins, post-execution verification, and finite-sample selective risk control.

We call the resulting approach *guarded agentic compaction* (GAC): a guarded *specialization* that compiles the routine part of a workflow — the recurrent read-only evidence prefix — while leaving every decision that the evidence does not determine to the model. Compaction names the mechanism, specialization names what it is allowed to do, and the boundary between them is the paper's subject. The shipped implementation contains two passes over a common typed trace representation: guarded region compilation (GRC), studied here, and trace-guided workflow specialization (TGWS), which routes entry states to smaller prompt/tool configurations. We isolate GRC because it changes execution semantics more directly and is the contribution with new real-scenario evidence.

Our research questions are:

RQ1. Can typed trace provenance reconstruct the dependencies needed to synthesize recurrent tool programs?

RQ2. On an unseen real-record workload, how does a learned compiled prefix trade factual task quality against requests, tokens, latency, and cost relative to an unchanged agent and to a hand-written composite tool?

RQ2b. Can a guarded compiler close the interface gap that comparison exposes, once a fair pre-model manual program and a learned prompt optimizer receive the same held-out workload?

RQ3. Does exact selective admission reject recurrent families when calibration support is inadequate?

RQ4. Which failure modes remain after structural compilation, and what does compaction *not* improve?

RQ5. Can group-level paired evidence select a risk-bounded optimization action before a fresh real-record cohort, and does that choice preserve the registered task contract while reducing resource use?

RQ6. Do the efficiency and preservation results transfer across distinct real-record workflow families with different tool vocabularies and output contracts?

The paper makes four contributions:

1. An evidence-licensed trace-to-program formulation that combines typed value provenance, effect and position barriers, bounded readable synthesis, empirical contracts, continuation-pinned composite projection, and runtime fallback.
2. A compile-or-retire admission protocol: the score and threshold grid are frozen before calibration, exact finite-sample bounds govern dispatch, and a framework-neutral portfolio can compare only transformations with paired evidence and compatible manifests.
3. A real-provider study spanning three public-record workflow families and 90 held-out records, plus a narrower five-repository time-forward extension on a frozen PR-outcome task. Together they separate execution conformance from source-grounded answer quality, preserve retained negative outcomes, and compare unchanged, compiled, and fair manual or template baselines without claiming superiority where they tie.
4. An identifier-aware empirical-contract refinement, motivated by an archived failed pilot, that treats opaque identifiers as typed/provenance-constrained values rather than unsafe numeric ranges. Revision-pinned NESTFUL and API-Bank experiments supply complementary refusal evidence when recurrent families lack admission support.

Alongside the primary live study, we add two narrower extensions. First, a frozen-source cross-repository GitHub checkpoint executes a simplified PR-outcome task across five repositories, reaches 580/580 exact discovery traces, completes four time-forward held-out repository cohorts exactly, and retires the fifth at compile time. Second, a bounded

public-record extension checkpoint records 420 privacy-modified HMDA groups that pass an independently implemented, provider-free exact gold reconstruction. HMDA is reported as a source-fidelity and control-plane preflight, not as a live optimization result; it is excluded from every effectiveness denominator and cross-domain efficiency claim.

We do *not* claim semantic equivalence, production certification, full-workflow cross-repository or time-forward generalization, or superiority to hand-written code. GCS synthesizes only a read-only view over an admitted program; it does not infer undeclared semantics or fuse physical reads. The six-case manual result is parity, and the bounded GEPA result does not generalize to prompt optimization as a whole.

Roadmap. Section 2 formalizes episodes, candidate regions, the position invariant, and the selective objective. Section 3 presents the method as a cascade of independent rejection points (fig. 1) and states the calibration guarantee. Section 4 defines the evidence tiers and the hypotheses they test, and section 5 answers RQ1–RQ6 (RQ2b with RQ2), opening with the hypothesis-by-hypothesis verdict in table 3 and closing with the five negative results and what each one rules out (section 5.11). Section 6 positions GAC against adjacent optimizers, and section 8 groups the threats to validity by the kind of inference each endangers. Every rejected candidate, the archived failed pilot, and the unsupported claims are retained rather than pruned, so the denominator of each result stays visible. The appendix carries what would otherwise interrupt that argument: the exact command behind every study (appendix A), the implementation audit (appendix B), the complete claim register including the thirteen claims the evidence does not support (appendix C), the per-metric numbers behind the figures (appendix D), the disposition of all ten benchmark families (appendix E), and the artifact lifecycle boundary (appendix F).

2 Problem Formulation

2.1 Executions and candidate regions

An episode $E = (z, M, e_{1:T}, y)$ contains an entry-state snapshot z, a content-addressed execution manifest M, ordered events e_t, and an observable outcome y. Events include model requests/responses, tool calls/results, handoffs, guardrails, approvals, errors, and commit boundaries. Each tool f has an application-owned effect declaration $\epsilon(f)$ and capabilities such as *speculatable* and *replayable*. Unknown effects are not reads.

A candidate region $R = [a, b]$ begins at a model-request boundary and ends after a tool result. It is *groundable* when every call argument has an expression over entry state or prior results within R:

$$\forall c_j \in R,\ \forall u \in \mathrm{args}(c_j), \quad u = g_{j,u}\big(z, o_{<j}\big), \qquad g_{j,u} \in \mathcal{L}, \tag{1}$$

where $\mathcal{L}$ is a closed, bounded transform library. It is *effect-admissible* when all tools are declared read-only, speculatable, replayable, and within the same principal/isolation partition. Handoffs, approvals, writes, errors, and unknown effects are barriers.

The current runtime resolves artifacts at the initial model boundary. Therefore the deployable compiler adds a position invariant:

$$a = 0 \quad \text{in the normalized model-boundary index.} \tag{2}$$

This restriction is not aesthetic. A suffix program dispatched at entry may reorder or duplicate earlier calls; section 5.10 demonstrates the failure empirically.

2.2 Selective optimization objective

The compiler emits an artifact $A = (P, H, V, q, \eta, M)$: deterministic program P, hard guard H, verifier V, nonconformity score q, threshold η, and manifest pins M. For a future episode, dispatch is selective:

$$d_A(z) = \mathbb{1}\{H(z) = 1 \wedge q(z) \le \eta \wedge M \text{ matches}\}. \tag{3}$$

If any precondition fails, the baseline agent runs. If execution fails before a commit and staging proves the attempt clean, the baseline agent runs. Dirty failure is an incident, not fallback.

Let $L(A, z) \in \{0, 1\}$ indicate a wrong compiled execution after dispatch and $C(A, z)$ its execution cost. We seek an artifact with high coverage and savings subject to bounded selective risk:

$$\max_{A} \quad \mathbb{E}\left[d_A(z)\{C(B, z) - C(A, z)\}\right] \tag{4}$$

$$\text{s.t.} \quad \Pr[L(A, z) = 1 \mid d_A(z) = 1] \le \alpha, \tag{5}$$

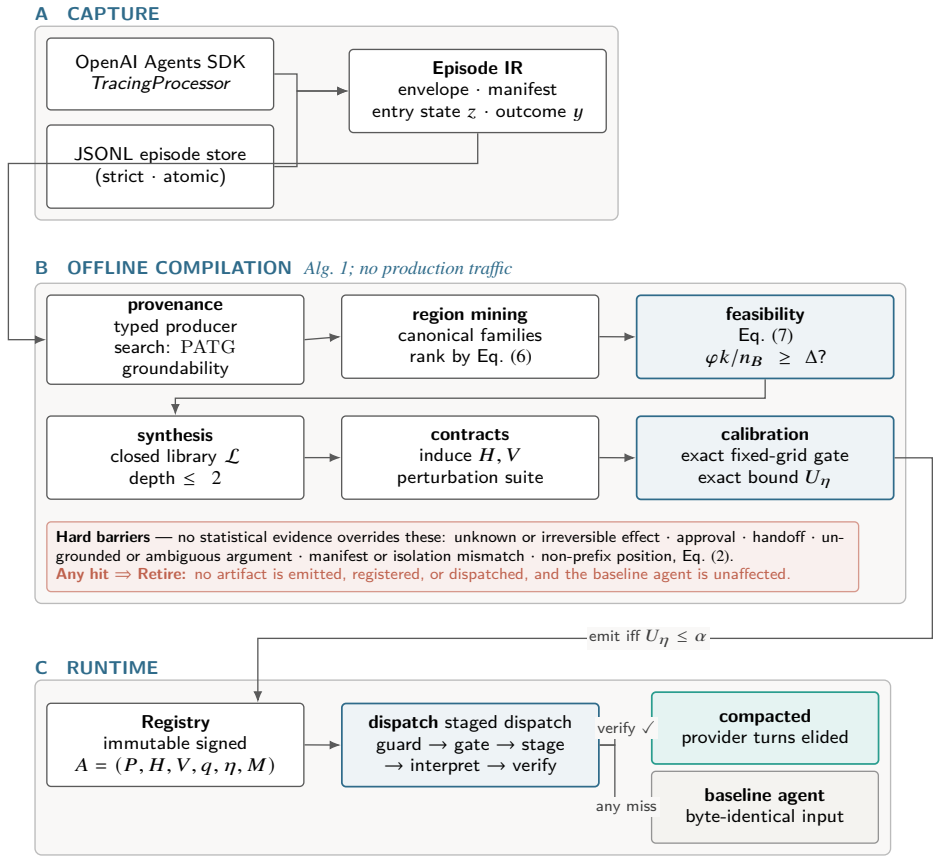

Figure 1: System architecture of GAC. **(A)** Capture normalizes framework traces into one typed Episode IR. **(B)** Offline compilation touches no production traffic and is a cascade of independent rejection points; the shaded *barrier* band lists conditions under which no statistical evidence can license an artifact. **(C)** Runtime resolves at the entry boundary. Of the terminal runtime edges, exactly one produces a compacted execution; five return the baseline and one raises an incident. "Baseline" is exact only where a staging owner holds the commit boundary: the outer runner can restore byte-identical model input, whereas the model-boundary adapter cannot un-emit a response the host has already committed to history (section 3.7.1).

where B is the unchanged agent. The implementation may return no artifact (RETIRE); this is a valid optimizer output.

At the compiler layer the implemented action set is $\{B, A\}$: retain the baseline or admit one compiled artifact. The separate portfolio layer in section 3.6 consumes paired measurements for arbitrary named actions. It does not weaken compiler admission or turn an unevaluated macro into deployable code.

Equation (5) is a design target, not a problem the implementation solves. Algorithm 1 ranks families by (6) and returns the *first* family that survives synthesis, contracts, and calibration; it neither enumerates all admissible families nor argues a bound on the gap to the best one. This matters more than a presentational note, because it makes the ranking load-bearing after all: since the first admissible family ships, the two coarse terms in (6) can change *which* artifact is registered even though neither participates in any admission decision. Reading ρ_{eff} from the effect catalog and evaluating all admissible families before selecting are both small changes, and both are future work.

3 Guarded Agentic Compaction

GAC is a cascade of independent rejection points (fig. 1). Algorithm 1 states that cascade in one place, because the order the stages run in is itself part of the argument: each stage can only weaken a candidate, never rescue one, and every RETIRE is recorded with the stage that caused it, so rejected candidates stay in the denominator. The rest of this section expands the three stages that carry the safety claim — provenance (Alg. 2), calibration (Alg. 3), and the runtime boundary (Alg. 4).

Algorithm 1 The end-to-end compilation cascade, from qualified traces to a registered artifact. Every **retire** is recorded with its stage so rejections stay in the denominator. Stages marked (CALLER) are separate library entry points rather than steps inside `compile_grc()`, which receives the split object and does not invoke the feasibility estimator itself; the cascade is presented in one place because that is the order in which the stages must run, not because one function performs all of them.

Input: episodes $\mathcal{E}$; effect catalog C; entry schema Θ; risk budget α; confidence budget δ; grid Λ
Output: artifact $A = (P, H, V, q, \eta, M)$, or RETIRE with an attributed reason

$\mathcal{E} \leftarrow \textsc{Qualify}(\mathcal{E}, C)$ ▷ drop partial runs, unpinned manifests, missing payloads
$\{\mathcal{E}_M\} \leftarrow \textsc{PartitionBy}(\mathcal{E}, \text{manifest, principal, isolation})$
for all partitions $\mathcal{E}_M$ **do**
  $(\mathcal{T}, \mathcal{D}, \mathcal{K}, \mathcal{S}) \leftarrow \textsc{GroupedSplit}(\mathcal{E}_M)$ (CALLER) ▷ train / dev / calibration / sealed test, split by *group*
  $G \leftarrow \textsc{BuildPatg}(\mathcal{T}, \Theta)$ ▷ typed provenance search
  $\mathcal{W} \leftarrow \textsc{MineRegions}(G, C, w_{\max})$
  $\mathcal{F} \leftarrow \textsc{Canonicalize}(\mathcal{W})$ ▷ hash signature, topology, run and live-in/out shape
  keep $F \in \mathcal{F}$ with cross-group support (and cross-day, if configured)
  if $\varphi\, k/n_B < \Delta$ **then** (CALLER)
    return RETIRE **(infeasible ceiling)** ▷ a separate estimator pass; run it before paying for synthesis
  end if
  for all families F ranked by score (6) **do**
    $P \leftarrow \textsc{Synthesize}(F, \mathcal{L})$ ▷ closed library, depth ≤ 2, group-wise refit
    if $P = \perp$ **then**
      continue (ungroundable slot)
    end if
    $(H, V) \leftarrow \textsc{InduceContract}(F)$
    if $\neg\textsc{ReplayAndChallenge}(P, V, \mathcal{D})$ **then** ▷ requires a sandbox; records `perturbations_claimed` otherwise
      continue (counterexample) ▷ stale reads, reordering, empty sets, drift, faults
    end if
    $q \leftarrow \textsc{FitScore}(\mathcal{D})$; **freeze** q ▷ dev groups only; never calibration
    $\eta \leftarrow \textsc{Calibrate}(q, \mathcal{K}, \Lambda, \alpha, \delta)$ ▷ fixed-grid exact admission
    if $\eta = \perp$ **then**
      continue (no admissible threshold)
    end if
    return $\textsc{Package}(P, H, V, q, \eta, M)$ with evidence and lifecycle
  end for
end for
return RETIRE

3.1 Typed provenance and canonical families

A capture adapter normalizes framework traces into the Episode IR. The OpenAI Agents SDK is a natural substrate because its agent loop records model calls, tools, agents, guardrails, and handoffs [10]; the compiler does not depend on the SDK. The artifact persists normalized episodes as canonical local JSONL with atomic snapshot replacement and strict validation. We intentionally removed an unused MLflow adapter: none of the experiments consumed its search or tracking surface, while retaining it added a second compatibility and privacy boundary. This simplifies reproduction but gives up server-side trace search and leaves framework neutrality demonstrated by the typed seam and dependency layering, not by two independently validated foreign-trace mappings. Manifests pin prompt, policy, guardrail, tools, model, SDK, tracer, entry contract, and effect-catalog identities. Episodes with different compatibility keys are partitioned before learning.

Above that seam, an optimization-pass API composes GRC and trace-guided workflow specialization without coupling either algorithm to the SDK. The evidence portfolio is a later decision layer: it consumes paired measurements for named actions and cannot synthesize, approve, or execute an unmeasured macro. This separation keeps trace capture, program construction, empirical action selection, and runtime lifecycle as independently auditable boundaries.

For each call-argument slot, the program-argument trace graph (PATG) searches entry-state and prior-result paths for typed values and bounded transforms. Exact matches, stable field paths, and transformations in $\mathcal{L}$ become candidate producer edges. Ambiguous candidate sets above a configured cap and ungrounded values block a region. This is execution provenance in the database sense [11], specialized to tool arguments rather than tuple derivations. Algorithm 2 states the search. Its two rejection branches carry the design commitment: a slot with no witness is a genuine model decision and a slot with too many witnesses is ambiguous, and both block the region instead of resolving to the cheapest available explanation.

Algorithm 2 BUILDPATG — typed argument provenance. A slot with no witness is a genuine model decision; a slot with too many witnesses is ambiguous. Both block the region rather than defaulting to the cheapest explanation.

Input: episodes $\mathcal{T}$; entry schema Θ; library $\mathcal{L}$; ambiguity cap κ
Output: provenance graph G with an edge per grounded argument slot

$G \leftarrow \emptyset$
for all episodes $E = (z, M, e_{1:T}, y) \in \mathcal{T}$ **do**
  $\Sigma \leftarrow \{(\text{“}z\text{”}, \pi, v) : (\pi, v) \in \textsc{Flatten}(z), \pi \in \Theta\}$ ▷ only allowlisted entry paths are eligible
  for all tool calls $c_j \in e_{1:T}$ in order **do**
    for all argument slots $u \in \text{args}(c_j)$ **do**
      $\mathcal{H} \leftarrow \{\, \sigma \in \Sigma : \exists\, g \in \mathcal{L},\ g(\sigma) = u,\ \text{depth}(g) \le 2 \,\}$
      if u is declared *literal-only* for c_j **then**
        $\mathcal{H} \leftarrow \{\sigma \in \mathcal{H} : g = \text{id or } g \text{ constant}\}$ ▷ never reconstruct a page index or a limit
      end if
      if $\mathcal{H} = \emptyset$ **then**
        mark c_j MODEL-ORIGINATED; **block** region
      else if $|\mathcal{H}| > \kappa$ **then**
        mark c_j AMBIGUOUS; **block** region
      else
        $G \leftarrow G \cup \{\sigma \xrightarrow{g} (c_j, u)\}$ for the minimum-description-length σ
      end if
    end for
    $\Sigma \leftarrow \Sigma \cup \{(\text{var}(c_j), \pi, v) : (\pi, v) \in \textsc{Flatten}(o_j)\}$ ▷ the result becomes a witness for later slots only
  end for
  for all pairs (e_s, e_t) sharing a resource where either writes it **do**
    $G \leftarrow G \cup \{e_s \prec e_t\}$ ▷ order edge: a reordering can never be proposed across a conflict
  end for
end for
return G

The miner enumerates model-boundary-to-result windows with at most $w_{\max}$ tool calls, qualifies effects and live-ins, and hashes the tool signatures, dependency topology, run shape, and live-in/out shape while abstracting literal values. Families require support across independent scenario groups rather than raw event frequency; the implementation also supports a minimum-distinct-days requirement, which the live study of section 5.3 does not exercise (it sets `min_days = 1` over a single-snapshot corpus). With N episodes, T events, and a bounded call window $w_{\max}$, enumeration is practically bounded by the number of reachable result boundaries; the implementation's nested scan is worst-case $O(NT^2)$ when non-call spans are adversarial. We report this rather than the stronger $O(NT)$ claim sometimes suggested by the bounded-call intuition.

Surviving families are ranked by a minimum-description-length trade-off that pays for expected savings and charges for variance, effect exposure, and program size:

$$\text{score}(F) = \underbrace{s_F\, \bar{k}_F\, c_m}_{\text{expected saving}} - \lambda_1 H(F) - \lambda_2 \rho_{\text{eff}}(F) - \lambda_3 |F|, \tag{6}$$

where s_F is scenario-group support, $\bar{k}_F$ the mean removable model requests, c_m their unit cost, and $H(F)$ the Shannon entropy over canonical variants inside the family. The penalties matter: without λ_1 the ranking prefers a heterogeneous family whose single canonical form fits none of its members.

Two terms are coarser in the implementation than the notation suggests, and we state them as implemented. ρ_{eff} is intended as the declared-effect exposure of the region, but the shipped ranking approximates it by the fraction of tool names containing a namespace separator — a naming convention, not a reading of the effect catalog. $|F|$ is intended as the size of the program the family would produce, but ranking precedes synthesis, so the implementation substitutes the argument-slot count of the first observed window. Both appear only in a *ranking* heuristic: no admission decision depends on either, and the effect catalog is consulted where it is load-bearing — in qualification and in the runtime facade. Making them faithful would reorder candidates and therefore require re-running the offline study; we list it as future work rather than describe semantics the code does not enforce.

Before any synthesis runs, a feasibility estimator bounds what the compiler could achieve even with a perfect verifier and a gate that never abstains:

$$\Delta_{\max} = \frac{\varphi\, k}{n_B}, \qquad \text{necessary:} \quad \varphi\, \rho\, k \ \ge\ \Delta\, n_B, \tag{7}$$

with n_B baseline model requests per episode, φ the fraction of episodes holding at least one eligible region, k the requests removed per successful dispatch, and ρ the verifier pass rate. Because (7) assumes $\rho = 1$, no measured reduction can *exceed* it, and it is met with equality exactly when nothing abstains and nothing fails. That is what happens in section 5.3: with $\varphi = 1$, $k = 3$ and $n_B = 4$ the ceiling is 0.750 and the measured reduction is 75.0%. A saturated ceiling carries no information about the compiler — it confirms only that the task fully determined the region, which for that workload it does. Reporting the ceiling first makes a decisive negative answer cheap: a workload whose ceiling is under the target can be declined without building a compiler for it.

3.2 Bounded synthesis and contracts

Program synthesis searches a 23-operator library with depth at most two. Operators cover path selection, indexing, string normalization, arithmetic, comparisons, bounded collection operations, and narrow loops. Candidate bindings are ranked by stability across groups; a group-wise refit rejects bindings that exploit row-level coincidence. The approach resembles programming-by-example [12] and bounded sketching [13], but the hypothesis space is deliberately small enough to inspect. It does not generate arbitrary Python.

Hard guards constrain entry types, required fields, categorical sets, numeric intervals, patterns, isolation keys, allowed effects, and manifest pins. Verifiers constrain live-out presence, type, cardinality, empirical hulls, provenance, effect multisets, and call counts. These are likely invariants [14], not proofs for all future inputs. Replay and perturbation are therefore available to challenge a candidate before calibration, but they are only as strong as the substrate supplied: without a sandbox the suite cannot run at all, and the artifact then records `perturbations_claimed: false` rather than implying a challenge occurred. The headline live artifact of section 5.3 is in exactly that position, and we report it there as a limitation of that run rather than as a property of the method.

3.3 Guarded composite synthesis

The macro comparison revealed a structural asymmetry: the manual baseline may redesign the application interface, while the original compiler preserves every source-tool observation. GCS closes that specific gap without moving application code into the optimizer's trust boundary. After ordinary synthesis and admission, it packages the same program as $\widetilde{P} = (P, \Pi, \tau_c)$, where P is the verified internal program, Π is a declared projection over verified live-outs, and τ_c is the exact compatibility key of the continuation that consumes the projection. A tool must carry an explicit `batchable` capability before it can appear inside this interface.

Some recurrent calls differ only in arguments that are irrelevant to the registered task. The effect catalog may therefore attach a signed, declarative task-semantic canonicalization to a dotted argument path. The implementation admits only five closed operations (integer clamping, whitespace stripping, case folding, stable set-like sorting, and finite aliases), and integer rules may declare an admissible input domain. This is not learned equivalence: values outside that domain raise and deoptimize. In the GitHub task, the consumer uses at most the first three comments and the snapshot tool already caps its result at three, so observed limits of at least three share the representative `limit=3`; `limit=1` is deliberately outside the contract.

The projection language reuses the bounded binding DSL. It can select only existing verified live-outs and cannot call arbitrary code, resolve a dynamic tool, or access unverified state. Runtime execution retains all internal source calls, effects, and field-level provenance. Verification and projection both complete inside the staging boundary; a missing projected field returns the baseline before release. A pre-model dispatch additionally requires an exact continuation-manifest match. It then exposes one native composite observation to the provider, so the provider receives the sufficient record on its first request rather than spending a request selecting the composite.

GCS is consequently narrower than automatic API fusion. It does not parallelize the internal calls, reduce the number of source reads, generate a new remote service, or prove the application-supplied task-semantic contract. Its contribution is a serializable, fail-closed interface transformation over a program the existing compiler can already verify.

3.4 Fair manual and learned-optimizer interfaces

To separate runtime placement from automatic discovery, we implement a second pre-model runner that does not consume a compiler artifact or its calibrated gate. It accepts an explicit, hand-authored read program, source and continuation manifests, effect catalog, bounded projection, and output/provenance/call-count verifier. Manifest drift, an undeclared effect, a missing field, a repeated observation, or any verifier failure returns the unchanged agent before release. This is a guarded laboratory baseline, not an independently reviewed production program; its construction cost is not measured.

Algorithm 3 CALIBRATE — fixed-grid exact selective admission. The grid and the score are frozen *before* this procedure sees a calibration group, which is what makes the union bound legitimate.

Input: frozen score q; calibration groups $\mathcal{K}$; grid Λ ($|\Lambda| = 11$); risk budget α; confidence budget δ
Output: threshold η with simultaneous guarantee, or $\perp$ (RETIRE)

$\mathcal{A} \leftarrow \emptyset$ ▷ admissible (coverage, threshold) pairs
for all $\eta \in \Lambda$ **do**
  $K_\eta \leftarrow \{\, g \in \mathcal{K} : \exists z \in g,\ q(z) \le \eta \ \wedge\ H(z) = 1 \,\}$ ▷ groups this threshold would admit
  $n_\eta \leftarrow |K_\eta|$
  $k_\eta \leftarrow |\{\, g \in K_\eta : g \text{ contains a violation} \,\}|$ ▷ *wrong after dispatch* only, never abstention
  if $n_\eta = 0$ **then**
    continue
  end if
  $U_\eta \leftarrow \mathrm{Beta}^{-1}\!\left(1 - \frac{\delta}{|\Lambda|};\ k_\eta + 1,\ n_\eta - k_\eta\right)$ ▷ Eq. (8); closed form $1 - (\delta/|\Lambda|)^{1/n_\eta}$ when $k_\eta = 0$
  if $U_\eta \le \alpha$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(n_\eta/|\mathcal{K}|,\ \eta)\}$
  end if
end for
if $\mathcal{A} = \emptyset$ **then**
  return $\perp$ ▷ with $k = 0$ this needs $n_\eta \ge \log(\delta/|\Lambda|)/\log(1-\alpha) \approx 92$
end if
return $\arg\max_{(\mathrm{cov},\eta)\in\mathcal{A}}\ \mathrm{cov}$ ▷ largest coverage among certified thresholds

For learned prompt optimization we wrap the official GEPA `optimize_anything` interface behind a provider-agnostic adapter. The adapter requires disjoint train and validation identifiers, enforces hard proposal, metric-call, and candidate-length budgets, records a sanitized audit log, and leaves tracking disabled. Only one operational-strategy sentence is mutable; the factual output schema, safety instructions, tools, grader, and quality-dominant objective remain fixed. Prompt optimization and guarded program execution therefore change orthogonal variables and can be evaluated alone or together. Optimization requests, tokens, latency, and cost are accounted separately from held-out evaluation.

3.5 Exact risk-gated admission

The score $q(z)$ is a logistic model over entry-observable features: unseen category, distance proxy, missing fields, hull margin, temporal drift, provenance ambiguity, and branch entropy. It is trained on development-group *unproductive* outcomes (wrong or abstained), then frozen. Safety calibration uses only *violations* (wrong after dispatch). The fixed threshold grid is

$$\Lambda = \{.02, .05, .08, .11, .14, .17, \\ .20, .25, .30, .40, .50\}.$$

For threshold η, let n_η calibration groups be admitted and k_η contain at least one violation, and write $\gamma = \delta/|\Lambda|$. The compiler computes the one-sided Clopper–Pearson upper bound [15]

$$U_\eta = \begin{cases} 1, & n_\eta = 0 \text{ or } k_\eta = n_\eta, \\ \mathrm{Beta}^{-1}(1-\gamma; k_\eta + 1, n_\eta - k_\eta), & 0 \le k_\eta < n_\eta. \end{cases} \tag{8}$$

For $k_\eta = 0 < n_\eta$, this reduces to $U_\eta = 1 - \gamma^{1/n_\eta}$. The largest-coverage admissible threshold with $U_\eta \le \alpha$ is selected; if none exists, the family retires. The explicit edge cases match the implementation and prevent an empty admitted set or an all-violation set from producing an undefined beta quantile. Algorithm 3 gives the selection loop. The ordering in it is the load-bearing part: the score and the grid are frozen before the procedure reads a single calibration group, which is what makes the union bound over Λ legitimate.

Proposition 1 (Per-candidate simultaneous calibration)**.** *Condition on the train/development data and fix one candidate program, hard guard, score q, group-level violation rule, and finite grid Λ before calibration. Let the calibration groups be i.i.d. from the same distribution as future groups. For each η, let A_i^η indicate admission, let Y_i^η be the potential group-level violation under dispatch, and define*

$$r_\eta = \Pr(Y^\eta = 1 \mid A^\eta = 1), \qquad n_\eta = \sum_i A_i^\eta, \qquad k_\eta = \sum_i A_i^\eta Y_i^\eta.$$

For a threshold with zero population admission probability set $r_\eta = 0$. Then, with probability at least $1 - \delta$ over calibration, $r_\eta \le U_\eta$ simultaneously for every $\eta \in \Lambda$. Consequently, the data-selected threshold $\hat{\eta}$ satisfies $r_{\hat{\eta}} \le \alpha$ whenever $U_{\hat{\eta}} \le \alpha$.

Table 1: Selective-risk configuration of every admitted artifact, generated from the sealed gates rather than restated in prose. These are per-candidate threshold-grid certificates, not candidate-search-adjusted compiler-wide guarantees. $\alpha = .05$ is the paper's registered target; three artifacts were calibrated at $\alpha = .10$ and do *not* meet it. All seven use $\delta = .10$ and the same 11-threshold grid, and all record zero calibration violations. Results resting on a row marked **no** are licensed only at the 10% selective-risk level.

Admitted artifact	Pairs	α	Admitted	U_η	$U_\eta \le .05$
Prescribed-prefix ablation	18	.05	92/92	0.0498	yes
Natural-order, three-read	18	.10	45/45	0.0992	**no**
Expanded replication (issue type)	30	.05	92/92	0.0498	yes
PR-outcome audit	30	.05	92/92	0.0498	yes
Backlog-attention routing	30	.05	92/92	0.0498	yes
Guarded composite synthesis	12	.10	88/92	0.0520	**no**
Comparator deployment (same GCS artifact)	6	.10	88/92	0.0520	**no**

Proof. Fix η. Conditional on $n_\eta = m > 0$, admitted groups are draws from the group distribution conditional on $A^\eta = 1$, so $k_\eta \mid n_\eta = m \sim \text{Binomial}(m, r_\eta)$. Inverting the exact one-sided binomial test gives $\Pr\{r_\eta > U_\eta \mid n_\eta = m\} \le \gamma$; for $m = 0$ the event is impossible because $U_\eta = 1$. Averaging over the random n_η therefore gives $\Pr\{r_\eta > U_\eta\} \le \gamma$. The union bound over the fixed grid yields

$$\Pr\{\exists \eta \in \Lambda : r_\eta > U_\eta\} \le |\Lambda|\gamma = \delta.$$

On the complementary simultaneous event the inequality holds for every grid point, hence also for the calibration-dependent choice $\hat{\eta}$. If $U_{\hat{\eta}} \le \alpha$, then $r_{\hat{\eta}} \le \alpha$. □

This proposition is deliberately *per fixed candidate*. The compiler implementation may calibrate more than one candidate family on the same groups and retain a surviving candidate after observing those calibration outcomes. Its current confidence budget is split across thresholds, not candidate families, so (8) alone does not provide a compiler-wide guarantee for that search. A direct Bonferroni repair for m candidates would use $\gamma = \delta/(m|\Lambda|)$; at $m = 2$, $\alpha = .05$, $\delta = .10$, and zero violations, the requirement rises from 92 to 106 admitted groups. Alternatively, the compiler must freeze one candidate before calibration or use a valid fixed-sequence procedure. The retained experiments are therefore reported with per-candidate conditional certificates, not a multiplicity-corrected compiler-wide certificate.

The guarantee concerns disagreement with the induced verifier V, not semantic task error. It also requires independent group indicators, no distribution shift, and a candidate fixed relative to calibration; clustered data or adaptive candidate/artifact selection needs a different risk allocation. The independence requirement is where our own live configuration is weakest and it is worth naming here rather than only in section 8: the live studies set `min_days = 1` and `min_principals = 1` over a single repository snapshot, so a "group" is one record from one project at one instant. Records drawn that way share a repository, a schema, an author community, and a period, and nothing in the calibration establishes that their violation indicators are independent. The bound is therefore conditional on a sampling assumption the snapshot cannot verify. For one fixed candidate, with $\alpha = .05$, $\delta = .10$, $|\Lambda| = 11$, and zero violations, 92 admitted groups are required. A single precommitted threshold would require 45, so the NESTFUL refusal is specific to the declared grid and risk configuration, not an intrinsic property of that benchmark. These boundaries are consistent with learn-then-test risk control [16, 17] and are revisited in section 8.

3.5.1 Which artifact was admitted at which risk level

α is a configured input, not a constant of the method, and this paper does not use one value throughout. Table 1 therefore reports the risk configuration of each admitted artifact next to the study it licenses. The three primary workflow families and the prescribed-prefix ablation record per-candidate calibration at the registered $\alpha = .05$ over 92 zero-violation groups. The earlier three-read natural-order artifact (section 5.3) and the guarded-composite artifact used by sections 5.7 and 5.8 were calibrated at $\alpha = .10$, with simultaneous upper bounds 0.0992 and 0.0520. Neither would have been admitted at .05, so every GCS result in this paper is a 10%-risk result and we mark it as such at each point of use rather than once in an appendix.

The GCS row is the informative one, because it fails the tighter bound for a reason worth stating. It is the only gate in the paper that rejects any calibration group at its selected threshold: it admits 88 of 92 (coverage 0.957) where every other artifact admits all of them. Rejecting four groups reduces n below the 92 that a zero-violation 5% bound requires, so the same discrimination that makes this gate less degenerate than the others is what puts its bound at 0.052. Coverage and risk trade off against each other exactly as the finite-sample calculation says they must, and at these sample sizes the trade is visible at the first sign of any selectivity at all.

3.6 Risk-bounded transformation portfolio

Compiler admission and application-level choice are distinct. A portfolio layer compares only actions measured on the same independent groups. It forms a declared weighted paired utility over cost, latency, tokens, and tool calls, then applies multiplicity-adjusted exact bounds to both task failure and non-positive utility. Among compatible actions passing both bounds and minimum support, it recommends the highest mean utility; otherwise it returns the baseline. Missing evidence is never imputed, manifest drift invalidates the decision, and macros remain review-required. This layer selects among measured actions; it does not synthesize application code or prove optimality over unmeasured alternatives.

3.7 Runtime and framework integration

At runtime, resolution is a bounded lookup: the registry indexes artifacts by compatibility and partition key, then filters the small candidate list at that key by lifecycle, kind, and signature. Cost is therefore constant in registry size but linear in the number of artifacts sharing one key, which the packaging step keeps small rather than the data structure guaranteeing it. The dispatcher checks lifecycle, manifest, hard guard, calibrated gate, mode, budget, and quota snapshot; executes through a permission facade into a staging area; verifies the result; and commits only on success. Shadow mode records would-dispatch behavior but does not change the agent. Live resolution accepts only active artifacts. The current version compiles reads only; irreversible operations remain with the baseline agent.

Algorithm 4 gives that boundary in full, and reading it against fig. 1 makes the asymmetry explicit: the cheap deterministic checks reject first, the calibrated gate runs only on what survives them, and of the terminal edges exactly one compacts. Five return the unmodified agent and one raises an incident, which happens only when an external commitment has already been made and rollback would be a fiction.

For the OpenAI Agents SDK, the library provides trace capture plus a custom model-boundary adapter for local function tools. Streaming, hosted tools, MCP tools, handoffs, loops, and assertions bypass rather than degrade. The SDK provides the loop and lifecycle, but it is not itself an offline optimizer [10].

3.7.1 Where "unmodified baseline" is exact, and where it is not

The two integration paths do not offer the same guarantee, and the difference is structural rather than an implementation gap. The outer runner owns the entry snapshot and the commit boundary, so a rejected attempt restores byte-identical model-visible input: for that path, "the baseline runs unchanged" is exact at every failure point. The model-boundary adapter cannot make the same promise. Once it has returned a synthesized `ModelResponse`, the host runner may already have committed that item to session history, and the `Model` interface provides no way to withdraw it; a verifier failure after that point delegates to the wrapped model with history the baseline would never have contained. Every check that can reject *before* emission — lifecycle, signature, manifest pins, hard guard, already-observed tools, quota attestation, and the calibrated gate — is therefore exact for both paths, and only post-emission verifier or binding failures are weaker for the adapter. Claims of universal clean fallback in this paper should be read as scoped to the staging-owning runner; the adapter is guarded substitution with explicitly weaker post-emission semantics. A general solution requires artifacts to target explicit control points with continuation tokens rather than implicit model boundaries, which we leave to future work (section 7.3).

4 Experimental Methodology

4.1 Evidence tiers and hypotheses

We use three non-overlapping evidence tiers, ordered by what each can license.

Tier 1 — external trace-compiler evidence. A sealed manifest pins NESTFUL and API-Bank, the two audited public corpora that retain the observed intermediate values needed to reconstruct and replay post-trace programs. They test provenance, synthesis, and admission rather than end-to-end agent quality. Eight additional benchmark adapters remain in the artifact's supplementary interoperability ledger, but not in the main evaluation: their checkers, task descriptions, simulated environments, hosted services, or missing outputs do not support a compiler-compatible head-to-head result.

Tier 2 — real records, live provider. Three primary GitHub workflow families use actual public records, deterministic local reads over one pinned snapshot, and live OpenAI provider calls. Each family has a distinct three-tool vocabulary and exact three-class output contract: issue type, pull-request outcome, or backlog attention. Every family uses 132 balanced discovery records and 30 disjoint balanced held-out records. The unchanged, compiled, and hand-written conditions receive the same records, model, factual contract, and condition placement. The latter two families com-

Algorithm 4 DISPATCH — boundary-time admission. Cheap deterministic checks reject first and the calibrated gate runs only on what survives them. Exactly one exit path compacts; the rest return the unmodified agent, and only a dirty post-commit failure raises an incident.

Input: entry state z; runtime context M'; registry $\mathcal{R}$; mode $m \in \{\mathsf{off}, \mathsf{shadow}, \mathsf{live}\}$
Output: observations for the region, or FALLBACK to the baseline agent B

if $m = \mathsf{off}$ **then**
  return FALLBACK ▷ conformance: model-visible input must be byte-identical
end if
$A \leftarrow \mathcal{R}$.RESOLVE(M'.compatibility, M'.partition) ▷ constant time
if $A = \bot \ \vee \ A$.lifecycle $\neq$ ACTIVE $\vee \ \neg$VERIFYSIGNATURE(A) **then**
  return FALLBACK
end if
if $H_A(z, M') \neq 1$ **then**
  return FALLBACK ▷ manifest pins, isolation keys, typed hulls, effect set
end if
if tools($A.P$) $\cap$ ALREADYOBSERVED $\neq \emptyset$ **then**
  return FALLBACK ▷ the region already ran; its live-ins are not the fitted ones
end if
if $m = \mathsf{live} \ \wedge \ \exists f \in$ tools($A.P$) : $C(f).\mathit{quota\_attested} \ \wedge$ SNAPSHOT $= \bot$ **then**
  return FALLBACK ▷ no staging owner, so a discarded attempt is not attestable
end if
if $q_A(z) > \eta_A$ **then**
  return FALLBACK ▷ calibrated abstention on an uncertain input
end if
if $m = \mathsf{shadow}$ **then**
  log would-dispatch; **return** FALLBACK ▷ observe coverage without changing behaviour
end if
$S \leftarrow$ STAGE.BEGIN()
$r \leftarrow$ INTERPRET($A.P$, z, FACADE(C)) ▷ bounded program; the facade re-checks effects at execution time
if r raised PRECOMMITERROR **then**
  assert S.REVERSIBLE(); **return** FALLBACK
else if r raised POSTCOMMITERROR **then**
  return INCIDENT ▷ an external commitment happened; do not feign rollback
end if
if $V_A(r) \neq 1$ **then**
  discard r; **return** FALLBACK ▷ live-outs, effect multiset and call count are checked before use
end if
S.COMMIT(r.effects); **return** r.observations

pile a verified three-read pre-model composite; the original issue family retains its conservative two-read prefix so its published negative comparison remains intact. These experiments establish transfer across workflow families on a real snapshot, not across repositories, time, or live GitHub service behavior.

Earlier fixed-prefix, depth, portfolio, GCS, and bounded-GEPA studies remain as ablations and negative controls. They preserve the historical denominator, but they are not counted as additional workflow families or pooled into the primary 90-record result.

Tier 3 — controlled stress suite. Six workflow *shapes* are run end to end through the real SDK runtime with live provider calls against deterministic local services. The business records here are fictional, so this tier cannot support a claim about real-world data; what it can do is cover control-flow structures the other two tiers do not contain at all — observation-dependent branches, pagination, a mandatory irreversible write, a handoff barrier, an undeclared-effect tool surface, and route specialization — and expose the behaviour of the runtime when it must *refuse*. We report it because a suite in which every case succeeds would be evidence of a weak test set, not a strong compiler. Three of its eight conditions are negative controls whose only correct outcome is no compaction.

4.2 Prospective public-record extension: HMDA

The extension harness adds a privacy-modified public Home Mortgage Disclosure Act (HMDA) loan-application-record (LAR) substrate to the same typed trace and effect contracts. Each case binds a year, filer identifier, row digest, non-sensitive field families, year-specific schema and code definitions, and source identifiers. Protected demographic attributes are excluded from the task and no output is interpreted as a lending, fairness, compliance, or legal decision.

Table 2: Provider-free public-record extension preflight. Exact gold is an independently implemented reconstruction, not a model-quality score. Variable paths identify cases that could support a future trace comparison.

Domain	Groups	Exact gold	Variable paths	Disposition
Vulnerability	420	420/420	48/420	Provider-free preflight
HMDA	420	420/420	416/420	Provider-free preflight; privacy-modified public LAR
SEC filing facts	0	–	–	Source-gated; no provider run

The primary oracle requires exact agreement with the frozen public row and its year-specific definitions; "NA", exempt, and unavailable states remain explicit values rather than imputed facts.

The provider-free validator reconstructs an independently generated gold record for all 420 HMDA groups (420/420 exact-gold passes). Of these, 416 groups (99.05%) contain a variable path suitable for a future trace study; the remaining four are valid fixed paths, not failures. The same preflight retains 420/420 vulnerability groups for context, while the SEC pool is unavailable because its compliant source contact is not configured. No OpenAI call, baseline-vs-GRC comparison, macro approval, token result, or latency result is licensed by this checkpoint.

The directional hypotheses below were fixed internally before the sealed test was scored, but the study was not externally preregistered: (H1) provenance places the expected producer in the candidate set for more than 90% of executable nested dependency slots — a recall statement, reported alongside unique-resolution and precision rates so it cannot be read as exact reconstruction; (H2) the compiled condition reduces provider requests and total tokens; (H3) no observed quality loss occurs on the sealed task; (H4) families below the exact-gate sample requirement retire; and (H5) a condition that must refuse reproduces the baseline model-call count and quality. H5 is deliberately *not* stated in dollars: section 5.4 shows refusing conditions costing more than baseline, for prompt-cache reasons unrelated to whether the refusal was correct. H3 is an observed-sample hypothesis, not a population equivalence claim. The portfolio pilot was designed after the replication was known but before its fresh cohort was selected or executed. Its prospective hypothesis (H6) is therefore narrower: the frozen selector's chosen action will pass all observed fresh task contracts and reduce its weighted resource objective relative to baseline. H6 was not externally preregistered and cannot establish that selection beats an always-macro policy. GCS was designed after both macro results were known. Its 12-pair study is therefore an explicitly post-study, exploratory test rather than an additional preregistered hypothesis. The issue selection is provider-outcome-free and disjoint from all 424 earlier issue IDs, but it was recorded by the same run that executed the provider calls rather than sealed in an externally timestamped protocol. The learned/manual comparator was likewise designed after the GCS result and is explicitly exploratory. Its test identifiers were frozen before prompt optimization, but the study was not externally preregistered and its six cases are not a powered non-inferiority test.

4.3 External benchmark: NESTFUL

NESTFUL contains more than 1,800 executable nested API sequences and was designed to test whether later calls correctly reuse earlier outputs [18]. We pin upstream commit `fc2c4123e735` (the full identifier is in the source manifest) and verify SHA-256 digests before use. Of 1,861 records, 1,416 use the audited `basic_functions` module; 1,415 execute successfully. One upstream record raises a string/integer type error and remains in the failure log.

We convert each successful execution into the same typed Episode IR, reconstruct expected producer references, mine families, split within families by group, synthesize from train/dev windows, and replay on held-out windows. This experiment does not call an LLM and does not measure NESTFUL planning accuracy. It tests the compiler after a valid trace exists. The exact gate is then applied to observed family support.

4.4 Three real-record workflow families

The primary transfer study reuses one revision-pinned 7,540-record public GitHub snapshot while changing the decision, tools, and exact grader. Pull requests are classified from record, merge-status, and discussion reads. Open non-PR issues are classified from record, ownership, and discussion reads. These tools are deliberately separate from the issue-type routing tools, preventing an apparent transfer result obtained by renaming one interface. Selection is provider-outcome-free, balanced by class, deduplicated by record identifier, and disjoint across discovery and test cohorts.

An initial pull-request pilot exposed two defects and is archived rather than overwritten: nullable merge timestamps were projected unsafely, and empirical numeric hulls treated opaque identifiers as quantities, rejecting unseen but schema-valid values outside the training extrema. The final compiler retains type, provenance, effect, and risk checks but uses an unconstrained value hull for fields named as identifiers, keys, or numbers; high- cardinality free text likewise avoids

an empirical literal regex. After this change, the pull-request final uses the original 132 discovery traces and a freshly sealed 30-record test cohort. The backlog cohort was frozen after the same implementation was fixed. Compiler rejection triggers the unchanged agent, so a failed guard cannot become a missing answer. Checkpoints are written before paid evaluation and after every condition.

4.5 Cross-repository, time-forward PR-outcome-core extension

To test whether the guarded pre-model runtime survives beyond one retained repository, we add a narrower cross-repository study over five frozen public GitHub snapshot sources from Hugging Face mirrors: `hugging-face/datasets`, `pandas-dev/pandas`, `psf/requests`, `streamlit/streamlit`, and `pytorch/pytorch`. The task asks for the exact pull-request record number and title, then classifies one of three exact outcomes: `open`, `merged`, or `closed_unmerged`. It uses two read-only local tools: record and merge-status. This removes comment-availability drift and keeps the exact grader repository-independent, but it is intentionally narrower than the three-tool workflow-family tasks.

Each repository first passes a provider-free preflight that deduplicates records, excludes earlier paper cohorts, seals 116 older discovery pull requests, and selects a balanced 30-record held-out cohort whose timestamps are strictly newer. The complete preflight supports 150 held-out pairs across the five repositories. Discovery then runs with the same provider, SDK, and model configuration as the earlier GitHub studies. Compilation freezes one candidate before calibration, uses 16 train, 8 development, and 92 calibration traces, and compares unchanged, compiled, and a fixed two-read template pre-model baseline on held-out records. Repositories with no admitted artifact are retained as fail-closed negatives rather than dropped.

Because that first five-repository cohort is class-skewed on some mirrors, we also run a balanced rerun on the three repositories with enough older records to satisfy the exact gate without changing the task: `pandas-dev/pandas`, `psf/requests`, and `pytorch/pytorch`. A provider-free round-robin selector seals 120 discovery pull requests and a disjoint 60-record held-out cohort per repository, each evenly split across `open`, `merged`, and `closed_unmerged` under the same strict time-forward rule. The model, frozen-candidate protocol, compiler, and three-arm held-out evaluation are unchanged. We treat this rerun as a protocol-sensitivity study: if it recovers open-state compaction or admits the repository that retired earlier, the earlier negative must be read as a cohort artifact rather than as an intrinsic limit of the two-read guarded runtime.

4.6 Controlled fixed-prefix live study

We pin revision `e344be7b84d1` (full identifier in the source manifest) of the Apache-2.0 `helmo/github-issues` dataset [19], verify the 12.7-MB Parquet digest, exclude pull requests, require at least 80 body characters, and deduplicate issue numbers. The scenario asks an agent to classify a `huggingface/datasets` issue as bug, enhancement, or question and return a short evidence-grounded summary. Three local function tools read the exact issue record, labels, and first three comments from the frozen public snapshot. The records and workload are real; the local tool service is deterministic so that API and website drift cannot contaminate paired comparisons.

Every model turn uses the OpenAI Agents SDK 0.19.2, OpenAI Python 2.52.0, and `gpt-5.6-luna` at low reasoning effort. The script loaded `OPENAI_API_KEY` for provider calls and `HF_TOKEN` for dataset retrieval; it serialized only boolean usage flags, never credentials. Cost is an estimate from the model-price table frozen with the experiment and the official pricing source [20].

The discovery cohort contains 132 issues, and all 132 satisfy the exact tool contract, so all 132 are compiler-eligible. The split consumes the first 116 under a fixed rule — train (16), development (8), calibration (92), sizes chosen before any outcome was scored, because $n = 92$ is fixed a priori by $(\alpha, \delta, |\Lambda|)$ — and leaves the remaining 16 unused. Those 16 are surplus to the calibration requirement, not filtered: they are neither excluded for a quality reason nor drawn upon, and we state the residue rather than report a denominator of 116 as though eligibility had produced it. A stronger design would either consume them or size the cohort to the split. The compiler, unmodified for the task, emits the prefix

$$\texttt{record}(z.id) \;\rightarrow\; \texttt{labels}(record.id) \;\rightarrow\; \texttt{comments}(record.id, 3).$$

It blocks all 116 suffix candidates under the position invariant. The 18-item sealed test contains six issues per category, is disjoint from discovery, and excludes all 162 issues touched by the earlier pilot. Baseline and compiled conditions run the same task and tools. Six cases are repeated once for determinism analysis.

The baseline requires four provider responses around three tool calls. The compiled condition executes the same necessary tools natively and asks the model once to produce the final structured response. Thus the intervention tests decision elision, not tool elision or parallel scheduling.

4.7 Natural-order, source-grounded live studies

The first follow-up keeps the pinned dataset, model, SDK, and three read tools but changes the task contract. Its prompt requests exact issue number, title, state, label-derived category, evidence label, and a verbatim excerpt from one of the first three comments; it contains no tool function name and no execution order. The grader checks the structured facts against the snapshot and normalized excerpt containment against the returned source comments. Tool order is reported separately and contributes nothing to task quality. Executable regressions verify that reordering calls leaves the score unchanged and that a fabricated comment excerpt fails.

We exclude all 312 records touched by the fixed-prefix final run or pilot. A stable hash selects 80 fresh discovery issues and 18 disjoint test issues (5 bug, 2 enhancement, 11 other; no class quota). All 80 discovery agents independently choose `record→labels→comments`, and all pass the corrected source-grounded contract. The executed artifact was built under the online oracle, which falsely rejected one short but valid excerpt: 79 records were then eligible, a stable hash assigned 20 train, 10 development, and 45 calibration records, and five were unused. Under the corrected oracle, issue 1741 would rank sixth and replace issue 3511 in that split. We neither retrain the executed artifact post hoc nor assume the two artifacts are equivalent; the result manifest records the consequence. No filter uses tool order. The review-driven protocol was checked into executable tests before provider execution but was not externally preregistered.

The paired test adds a hand-written `issue_get_bundle` tool that returns the same record, labels, and three comments in one read. Each issue runs the unchanged agent, learned compiler, and macro exactly once. The six permutations of condition order are cycled three times across the 18 records, so every condition occupies every ordinal position equally often. All conditions use the same model and factual output schema. The full run made actual provider calls and records an estimated total cost of $0.0920; credentials are represented only by boolean usage flags.

The expanded replication uses the same provider and snapshot but a stricter typed output: exact issue number, title, state, total comment count, label-derived category and evidence label, plus a 20–240 character verbatim excerpt from the returned title, body, or comment payload. A provider-free preflight sealed 132 discovery records and 30 held-out records, balanced as ten bugs, ten enhancements, and ten questions. The paid run used those exact records and cycles all six baseline/compiler/macro orders five times. Ten test records are repeated once per condition for determinism.

Of the 132 discovery outputs, 130 pass the exact-source task and are eligible under the executed safety rule. All 132 read the record, then labels, then comments; however, the natural caller chooses different integer comment limits. Because no trace-grounded expression can reproduce that argument, the three-read candidate fails with `ungroundable_slot`. The compiler correctly falls back to the longest groundable prefix, `record→labels`, and the ordinary agent performs the comments read and final rendering. Sixteen training, eight development, and 92 calibration records are selected from the 130 eligible traces; the other 14 are unused. The complete run retains 252 agent outputs, 848 provider responses, no infrastructure failure, and an estimated total cost of $0.1913. An online literal-argument grader was corrected provider-free: all original outputs and measurements remain bound to the paid discovery checkpoint, while 212 prior quality objects are preserved under `online_quality`.

4.8 Prospective portfolio pilot

The portfolio calibration consumes only the 30 independent primary issue groups from the expanded replication. Compiler and macro observations are paired with the unchanged arm; ten repeated executions are averaged within their issue and do not increase support. We fix $U_{\min} = 0$, $n_{\min} = 30$, $\alpha_q = \alpha_u = .15$, and 95% familywise confidence split over two actions and two exact bounds. This 15% pilot limit is intentionally looser than the compiler's registered 5% contract bound and must not be conflated with it; nor should it be conflated with the 10% level at which the GCS artifact was calibrated (table 1). Three risk levels appear in this paper and each licenses a different set of results.

Before any prospective call, the script stores the decision and hashes the calibration file. A stable hash then selects 12 previously unused public issues: four bugs, four enhancements, and four "other" records. All eligible question records had already been consumed by earlier sealed protocols, so reusing them would have violated the prospective boundary. Each issue runs baseline and the selected reviewed action once; order is counterbalanced six each way. The non-selected action is not executed on this cohort. The provider-backed script requires an explicit macro-review flag, records 24 unique native traces, and serializes no credential. This evaluates a pre-data action choice on a fresh cohort, but only within one workflow family and one model configuration.

4.9 Bounded learned and pre-model comparator study

We first reconstruct both GCS and the manual program on all 12 selected split records without a provider call. Their projected evidence must match byte-for-byte before the paid study may proceed. Four records train prompt proposals, two evaluate them, and six remain sealed for held-out evaluation; all are disjoint from 443 issue identifiers used by

earlier experiments. Strict exclusion leaves bug, enhancement, and other records but no question record, so the held-out cohort balances only the three available categories (two each).

Official GEPA 0.1.4 may make at most 16 task-metric calls and three proposals. Each task evaluation is a real Agents SDK execution, and each reflection is a real Responses API call. The score

$$1000I_{\text{exact}} + 100q_{\text{facts}} - 5N_{\text{req}} - N_{\text{tool}} - N_{\text{tok}}/10{,}000$$

makes exact factuality dominant over the available efficiency terms. After optimization, all five conditions run once per held-out issue under a balanced condition-order schedule. Because GEPA retains its seed, the nominal GCS+GEPA arm is an order-balanced replication of GCS, not evidence of a distinct combined optimizer.

4.10 Controlled stress suite: workflow shapes

Six SDK-backed stress scenarios cover a linear prefix, permission-scoped retrieval, a handoff barrier, undeclared MCP effects, an observation-dependent branch ending in an irreversible write, and route specialization. Three negative controls require exact fallback. Their purpose is runtime boundary coverage, not domain evidence; complete scenario definitions and measurements are retained in the repository.

4.11 Metrics and statistical analysis

The primary efficiency outcome is provider requests. Secondary outcomes are input, output, and total tokens; provider and wall latency; estimated cost; and tool calls. In the fixed-prefix ablation, quality requires the correct category, evidence label, issue number, valid summary shape, and exact tool contract. In the natural-order studies, quality instead requires exact source-grounded fields and label decisions and is independent of tool order. The expanded task contract additionally requires one safe, grounded call to each necessary read tool while allowing any order and any integer comment-body limit. Paired differences use the same issues. We report 10,000-sample paired bootstrap confidence intervals [21] and two-sided Wilcoxon signed-rank tests [22]; binary quality uses exact McNemar tests [23]. Signed-rank p-values use the exact distribution where no ties occur and a tie-corrected normal approximation otherwise. One row is deliberately *not* given a p-value: every pair changes provider requests by exactly -3, so the difference is deterministic and a rank test on it reports an artifact of degeneracy rather than evidence. Secondary p-values are descriptive and unadjusted. Determinism is reported as decision, tool-trace, and byte-exact answer agreement. Peak memory and runtime are measured for the provider-free NESTFUL compiler stage. For portfolio admission, quality failures and non-positive group utilities are separate binary endpoints. Their one-sided exact bounds split 5% error across two actions and two endpoints; mean utility ranks only actions that pass both gates. The 12-record prospective cohort is an outcome evaluation, not additional calibration evidence. Its paired bootstrap intervals and signed-rank tests are descriptive and unadjusted. The exploratory GCS comparison uses the same paired procedures. Its primary contrast is GCS minus the measured provider-visible macro; it reports exposed tool interfaces and internal source reads separately so interface fusion cannot masquerade as eliminated work. The bounded comparator uses the same exact quality and paired resource procedures. Host monotonic wall time is primary. Provider-span latency is retained but excluded from a comparison if the summed spans exceed the surrounding host wall time by more than 250 ms. Optimization requests, tokens, latency, and cost are never mixed with held-out evaluation metrics.

5 Results

The evidence separates three questions. First, can a trace be reconstructed and synthesized? Second, is there enough independent evidence to admit the result? Third, if admitted, does it improve the end-to-end workflow against a strong manual alternative? NESTFUL and API-Bank answer the second question with refusal (section 5.1). The three GitHub workflow families answer the third within one repository, and the later cross-repository extension tests the same question on a simpler time-forward task (sections 5.2, 5.2.1 and 5.6). The remaining sections retain the earlier depth failure, failed suffix pilot, harder workflow shapes, and prospective portfolio study because each constrains a different claim rather than adding another headline.

Table 3 adjudicates every hypothesis before the detail, including the two that split and the two questions that were asked after the results were known and can therefore only be exploratory. Two of the six directional hypotheses are supported on one reading and not on another; recording the split is the point, because either half alone would misdescribe the result.

Table 3: Every hypothesis and its outcome. "Pre-specified" means fixed internally before the sealed test was scored; no part of this study was externally preregistered. RQ2b and the determinism question were formulated after the results they examine were known, so they carry no hypothesis and license no confirmatory claim.

	Question	Pre-specified	Outcome	Reported in
H1	RQ1	yes	Split: supported on recall (96.3% of dependency slots), not supported on unique resolution (80.7%) against a 90% target	section 5.1
H2	RQ2, RQ6	yes	Supported: requests −50.0 to −75.0% and total tokens −39.5 to −81.4% across three families	section 5.2
H3	RQ2	yes	Split: supported for the expanded two-read artifact (30/30 exact), not supported for the earlier three-read artifact (17/18)	section 5.3
H4	RQ3	yes	Supported: every synthesized family retires — 12 on NESTFUL and both on API-Bank — for insufficient calibration support	section 5.1
H5	RQ4	yes	Supported on model calls and quality for all three negative controls; deliberately not stated in cost, which rises up to 55.4%	section 5.4
H6	RQ5	before the fresh cohort	Supported on 12 fresh pairs (12/12 exact, requests −50.0%); cannot establish that selection beats an always-macro policy	section 5.9
—	RQ2b	no (exploratory)	GCS beats the measured provider-visible macro on one family, and reaches parity with a fairly placed manual program	sections 5.6 and 5.7
—	RQ4	no (post hoc)	Not supported: byte-exact answer agreement is 1/6 for the baseline and 0/6 compiled	section 5.10

Table 4: External NESTFUL compiler results. "Wrong" means a synthesized program executed but disagreed with the recorded held-out outputs; abstentions are explicit.

Measurement	Result
Executable basic-function episodes	1415
Expected producer in candidate set (recall)	5531/5746 (96.3%)
of which uniquely resolved	4636 (80.7%)
of which ambiguous (truth among many)	895 (15.6%)
Slots with no candidate	215
Candidate-edge precision	5531/6564 (84.3%)
Complete groundable windows	1207/1415 (85.3%)
Recurring families with support ≥ 5	32
Synthesized families	12/32
Held-out replay: pass / abstain / wrong	24 / 12 / 0
Maximum family support / gate minimum	26 / 92
Certifiable families	0
Peak memory / elapsed time	23.3 MiB / 5.98 s

5.1 RQ1 and RQ3: provenance succeeds more often than certification

Across 1,415 executable episodes, GAC places the expected producer in the candidate set for 5,531 of 5,746 dependency slots (96.3%). That figure is *candidate recall*, not dependency reconstruction, and it is close to definitional: candidates are generated by value matching and the gold producer emitted the matched value, so it joins the candidate

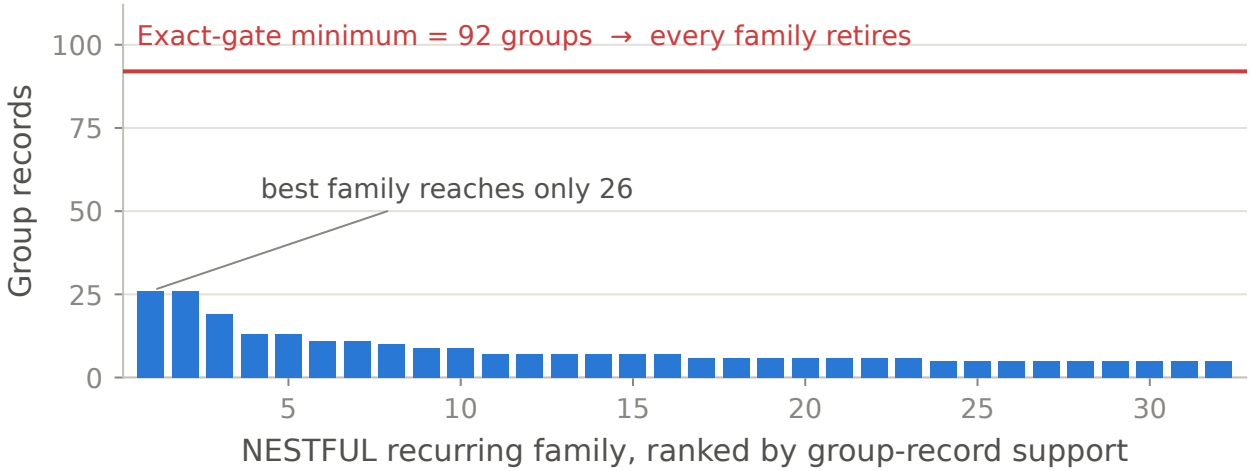

Figure 2: All NESTFUL families fall below the 92-group exact-gate requirement; the GitHub study was deliberately sized to reach it.

set whenever any admissible path exists at all. Consistently, there is not one slot in which a non-empty candidate set omitted the true producer — the 215 misses are exactly the 215 slots with no candidate. Recall here therefore measures search *reachability* and is insensitive to ranking quality; it cannot fall below the groundability rate however poor the ranking. The load-bearing numbers are the other two: only 4,636 slots (80.7%) are resolved to a unique producer, 895 (15.6%) contain the expected producer among several candidates, and 215 have no candidate at all. Counting all emitted candidate edges gives 84.3% precision (5,531 of 6,564). H1 was pre-specified internally as recovery above 90%; it is supported on recall and *not* supported on unique resolution, and we report both rather than the more favourable one. It finds complete groundable windows in 1,207 episodes (85.3%). The remaining full-window failures are overwhelmingly ambiguous value matches (207), with one ungrounded slot. That count is per *episode* under first-hit attribution, and it is not the same denominator as the 215 no-candidate *slots* in table 4: an episode is attributed to the first reason that blocks it, so an episode containing both an ambiguous slot and an ungrounded one is counted as ambiguous, and slots outside every enumerated window are counted in neither. The two numbers are therefore consistent but not comparable, and we report both rather than the one that reads better. Recurrence is fragmented: 714 compiler families exist, only 32 have support at least five, and maximum support is 26.

Of 32 attempted recurrent families, 12 synthesize. Their 36 held-out windows yield 24 passes, 12 abstentions, and zero wrong executions. The dominant synthesis failures are unsupported loop predicates and slots that fail grouped refitting. This zero-wrong result does not certify the programs: the per-candidate exact gate requires 92 zero-violation groups, so all families retire. H4 is supported, and the negative result distinguishes a guarded compiler from a recurrence-only macro miner.

Two properties of this benchmark path bound how far it generalizes. Its per-family split is lexicographic by episode identifier rather than randomized or chronological, so it tests held-out *windows* but not robustness to a shifted distribution; and the development partition it allocates is reported but not consumed, because synthesis reads the training windows and replay reads the test windows directly. This path also builds provenance, mining, synthesis, and replay itself rather than invoking the full compiler, so it is an executable structural benchmark, not end-to-end validation of the deployed gate.

The provider-free NESTFUL analysis completes in about six seconds, including roughly 1.4 for provenance construction, with 23.3 MiB peak traced memory. These numbers characterize this pinned dataset and Python process, not distributed production throughput.

5.2 RQ6: transfer across three workflow families

The result transfers across all three decisions and tool vocabularies. Compiled programs reach 90/90 exact outcomes versus 89/90 for unchanged agents. Aggregating raw paired counts, compilation reduces provider requests 66.6%, visible tool interfaces 44.2%, tokens 63.1%, observed wall latency 64.2%, and estimated cost 58.7%. Per-family ranges are more informative than the aggregate and are plotted in fig. 3: requests fall 50.0–75.0%, tokens 39.5–81.4%, latency 51.7–73.0%, and cost 32.0–75.3%. The single baseline miss in backlog routing does not establish a quality improvement

Table 5: Primary real-record, live-provider results. Entries are reductions from the unchanged baseline; exact counts are paired held-out task contracts. "Interfaces" counts provider-visible tool calls, not the three verified reads inside a composite.

Family	Exact $B \to C$	Requests	Interfaces	Tokens	Latency	Cost
Issue-type routing	30/30→30/30	50.0%	0.0%	39.5%	51.7%	32.0%
PR-outcome audit	30/30→30/30	75.0%	66.7%	80.8%	73.0%	75.3%
Backlog-attention routing	29/30→30/30	74.8%	66.3%	81.4%	68.9%	75.1%
Weighted total	89/90→90/90	66.6%	44.2%	63.1%	64.2%	58.7%

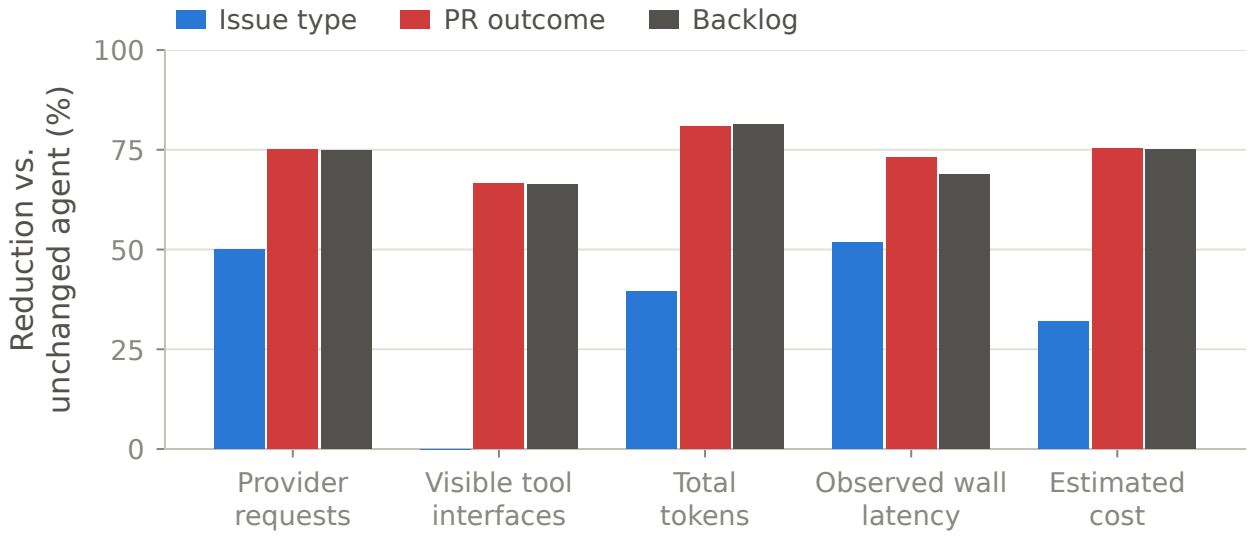

Figure 3: Per-family reduction against the unchanged agent. The issue-type family is the low end of every metric because its groundability proof admits only a two-read prefix and leaves the third read — and the interface — with the agent.

(exact McNemar $p = 1$). All three artifacts record the paper's tightest per-candidate calibration: registered $\alpha = .05$ over 92 zero-violation groups (table 1). This does not upgrade them to a candidate-search-adjusted compiler-wide guarantee.

Because the three arms are paired on the same records, the preservation statement can be pooled: across all 90 held-out pairs there is no compiled-only failure, so the one-sided 95% exact upper bound on the compiled-only discordance rate is 3.3%, against 9.5% from any single 30-pair family. This is the strongest preservation bound the paper's data supports. It is nonetheless a bound over three *different* admitted artifacts on one snapshot, so it constrains the observed sample rather than certifying any one artifact for a new workload, and it is still an order of magnitude away from a non-inferiority margin a deployment would want.

The strongest comparator prevents an exaggerated conclusion. Hand-written conditions also reach 90/90. On the two new families, fairly placed pre-model manual programs have the same single-request and single-interface structure as compiled programs and comparable resource use. The measured value of compilation is therefore automatic trace-grounded discovery, rejection, and lifecycle management; this experiment does not show runtime dominance over a correct manual implementation. All records come from one snapshot of one repository, so RQ6 is supported for workflow-family transfer only, not cross-repository or time-forward generalization.

5.2.1 Cross-repository, time-forward extension

A separate frozen-source extension asks a narrower question: can the guarded compiler survive beyond one repository when the task is exact and two-read? The provider-free preflight seals five repositories and a 150-case held-out capacity.

Table 6: Expanded natural-order Tier-2 replication on 30 fresh public issue records with live provider calls and all six condition orders balanced five times. Means use primary pairs only. Exact pass checks source-grounded fields; every arm also passes the safe-read task contract 30/30.

Condition	Requests	Tools	Tokens	Wall (s)	Cost	Exact pass
Unchanged agent	4.0	3.0	4259.4	6.16	$0.000872	30/30
GAC (two-read prefix)	2.0	3.0	2576.5	2.97	$0.000593	30/30
Hand-written macro	2.0	1.0	1780.8	3.25	$0.000545	30/30

The executed discovery pass reaches 580/580 exact traces (116/116 per repository). Four repositories admit artifacts and complete 120 held-out paired records: `huggingface/datasets`, `pandas-dev/pandas`, `psf/requests`, and `streamlit/streamlit`. Baseline, compiled, and fixed-template conditions all pass 120/120 exact contracts on those completed repositories.

Relative to baseline, compiled execution reduces provider requests 44.4%, total tokens 52.4%, observed wall latency 49.4%, and estimated cost 48.6% across the 120 held-out pairs. The fixed template is more efficient still, reducing requests 66.7%, total tokens 78.6%, observed wall latency 68.1%, and estimated cost 73.3%. That matters because it sharpens the real claim: on one conservative frozen cohort the learned artifact is valuable for automatic discovery, guarded admission, and retirement, not because it dominates a fixed ungated template on this simplified task.

This conservative cohort is informative because it retains a principled negative. Across the four completed repositories, the learned artifact compacts all 40 `merged` and all 40 `closed_unmerged` held-out pull requests, falls back on all 40 `open` ones, and `pytorch/pytorch` retires at compile time under the frozen-candidate exact gate.

A second rerun asks whether that open-only coverage pattern is intrinsic. On the three repositories with enough class support for a balanced strict time-forward split (`pandas-dev/pandas`, `psf/requests`, and `pytorch/pytorch`), a provider-free round-robin preflight seals 120 discovery pull requests and 60 held-out pull requests per repository, evenly split across `open`, `merged`, and `closed_unmerged`. The executed discovery reaches 360/360 exact traces. All three repositories admit the same two-read artifact, and baseline, compiled, and fixed-template conditions all pass 180/180 exact contracts on the held-out cohort. Relative to baseline, compiled execution reduces provider requests 66.7%, total tokens 78.4%, observed wall latency 60.7%, and estimated cost 72.7%. The fixed template is essentially tied on this simplified task: it matches 180/180 exact contracts and the same 66.7% request reduction, with slightly lower observed wall latency (63.0%) and slightly smaller token and cost savings (78.4% and 72.3%) than the learned artifact.

Crucially, the learned artifact now compacts all 60 `open`, 60 `merged`, and 60 `closed_unmerged` held-out pull requests, and the verifier's `pr.state` hull contains both `open` and `closed` on all three repositories. The earlier open-only fallback pattern and `pytorch/pytorch` retirement are therefore properties of the original frozen cohort design, not intrinsic limits of the two-read guarded runtime. The balanced rerun strengthens the paper's cross-repository evidence, but it does not widen the task: both extensions remain narrower than the three-tool workflow-family studies.

Independent refusal on API-Bank. API-Bank contains 389 observed calls over 49 tools. Conservative effect barriers leave 48 candidate windows in 37 tasks and 19 families; maximum support is eight. Two families synthesize, but their two held-out windows produce zero passes, two abstentions, and zero wrong executions. The configured exact gate requires 92 independent zero-violation groups and retires every family. Re-executing the pinned APIs reproduces 338 of 389 recorded calls and 162 of 212 complete tasks; fixture drift and missing dependencies are reported separately. Thus a second trace-complete corpus supports the same narrower result as NESTFUL: recurrence and replayability do not supply admission evidence.

5.3 RQ2: real-provider efficiency and observed quality

5.3.1 Natural tool ordering, compilation depth, and a live macro.

In the expanded replication, all 132 discovery traces read the record, then labels, then comments without the prompt naming tools or order; 130 pass the exact-source task. The compiler does *not* blindly turn recurrence into a macro. It rejects the full three-read candidate because the comments limit has no consistent trace-grounded expression, then emits the groundable two-read prefix. The gate admits 92/92 calibration groups with zero registered violations at $\alpha = .05$, $\delta = .10$, and 11 thresholds, for simultaneous upper bound 0.0498. Thresholds below 0.11 admit none and those at or above 0.11 admit all, so this remains a sample-size gate rather than a demonstrated risk–coverage discriminator.

Table 7: The earlier free-order 18-record follow-up, in which GAC compiled the more aggressive three-read region. Means are per issue over 18 held-out pairs with all six condition orders balanced three times. This is the artifact calibrated at $\alpha = .10$, not one of the primary families, and it is the arm that records the single factual miss discussed below.

Condition	Requests	Tools	Tokens	Wall (s)	Cost	Factual pass
Unchanged agent	4.0	3.0	4065.4	5.69	$0.000787	18/18
GAC	1.0	3.0	1381.0	1.34	$0.000404	17/18
Hand-written macro	2.0	1.0	1604.9	4.68	$0.000459	18/18

Relative to the unchanged agent, the partial compiler reduces requests 50.0%, total tokens 39.5%, observed wall latency 51.7%, and estimated cost 32.0%, while leaving three necessary reads. The macro also halves requests, but reduces tokens 58.2%, cost 37.5%, and tool calls 66.7%. It therefore beats the learned artifact on tools, tokens, and dollars; the compiler has lower observed mean wall time (2.97 versus 3.25 seconds), but its paired 95% interval against the macro crosses zero. Every primary arm passes 30/30 exact factual and full task contracts. With zero compiler-only failures, the one-sided 95% exact upper bound on the sample's underlying discordance rate is still 9.5%; this is observed preservation on one domain, not equivalence or non-inferiority.

Ten records are repeated once per arm. Category decisions agree in 10/10 for all arms, but byte-exact answers agree in 7/10 baseline, 6/10 compiled, and 4/10 macro pairs; tool traces agree in 9/10, 9/10, and 10/10. Compaction therefore does not improve free-text determinism here. The provider-free semantic regrade changes no output, token, timing, or cost measurement: it removes a planted literal comment-limit requirement that contradicted the "as needed" prompt, preserves all 212 online quality objects, and binds the final result to the paid discovery checkpoint by SHA-256.

The earlier 18-record follow-up tested the more aggressive three-read artifact and remains important negative evidence. Its artifact was calibrated at $\alpha = .10$ over 45 groups (upper bound 0.0992), so it too sits outside the registered 5% level (table 1); the factual miss below is therefore a miss by an artifact admitted under a weaker guarantee than the primary families receive.

Relative to its unchanged agent, that artifact reduces requests 75.0%, total tokens 66.0%, observed wall latency 76.4%, and estimated cost 48.7%; its macro reduces the same quantities by 50.0%, 60.5%, 17.7%, and 41.7% (table 7). The unchanged agent and macro pass 18/18 factual contracts, but GAC passes 17/18. On issue 6602 the source contains a Markdown link while the compiled continuation drops the URL and returns only its anchor text. The single discordance gives two-sided exact McNemar $p = 1$ and a one-sided 95% exact upper bound of 23.8%. H3 is therefore supported as an observed-sample result for the expanded two-read artifact and not supported for the earlier three-read artifact; preservation is not invariant to compilation depth.

The miss fixes the gate's scope. Zero calibration violations concern the compiled *tool program*: grounded arguments, tool outputs, call counts, and declared effects. The verifier does not certify the downstream model continuation. Thus 45/45 clean program replays and a held-out factual-output miss are compatible. A deployment claim requires a second admission layer over continuation outcomes or a deterministic checked renderer.

We implemented that boundary as a framework-neutral `ContinuationGuard`. It checks the candidate answer, then tries an optional deterministic renderer and baseline in that order, revalidating either output before release; invalid recovery returns no output. A provider-free replay accepts 17 retained compiled answers, detects issue 6602, and checked-renders it from source observations, yielding 18/18. This post-hoc mechanism check establishes detection and rendering on retained records, not live latency, cost, non-inferiority, or cross-domain safety.

That earlier study's online grader also imposed an undocumented excerpt minimum and misread a literal comment "none" as a sentinel. Its separate provider-free revision corrected four rows, retained online labels and the prior digest, and changed no provider metric. It did not retrain the executed artifact; after correction issue 6602 remains the only compiled-only failure.

5.3.2 Prescribed-prefix ablation.

The compiled GitHub artifact has 16/16 training-group support, passes all eight development replays, and admits 92/92 calibration groups with zero observed violations. At $\alpha = .05$, $\delta = .10$, and 11 thresholds, the simultaneous upper bound is 0.0498. The artifact was promoted only under a paper-protocol lab flag; no production canary is claimed.

Provider requests fall from exactly 4 to 1 per issue (75.0%; paired 95% bootstrap difference $[-3, -3]$). Mean total tokens fall 65.7%, observed wall latency 85.0%, and estimated cost 52.6%; tool calls remain three in both arms (fig. 4). The effect is not carried by outliers: every one of the 18 pairs falls below the no-change diagonal on tokens, latency, and cost

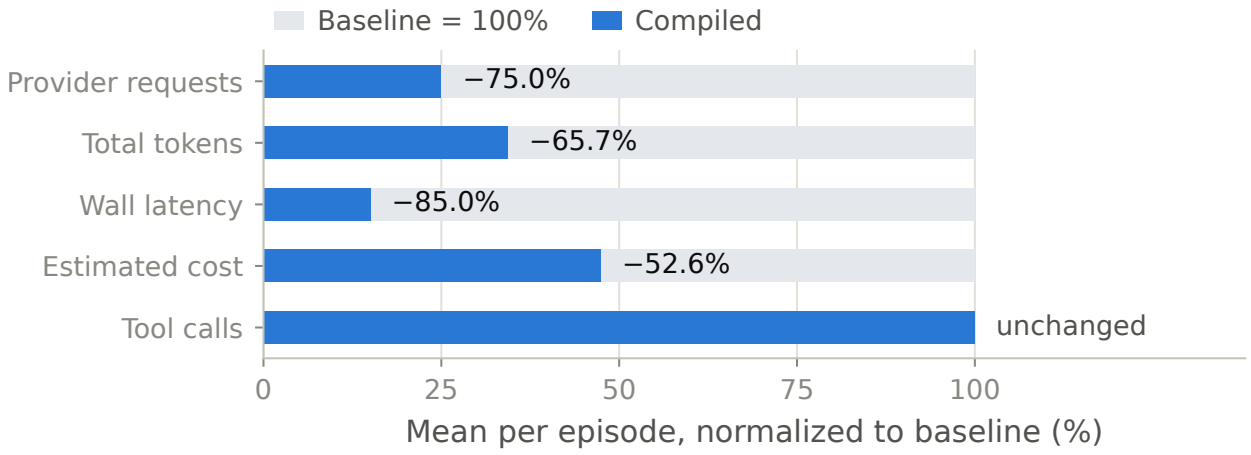

Figure 4: Prescribed-prefix ablation, 18 held-out pairs, each mean normalized to its own unchanged baseline. Tool calls are the control: the compiled program performs the same three reads, so what falls is the model-mediated control flow around them, not the work. Per-metric absolute means and Wilcoxon p-values are in table 14.

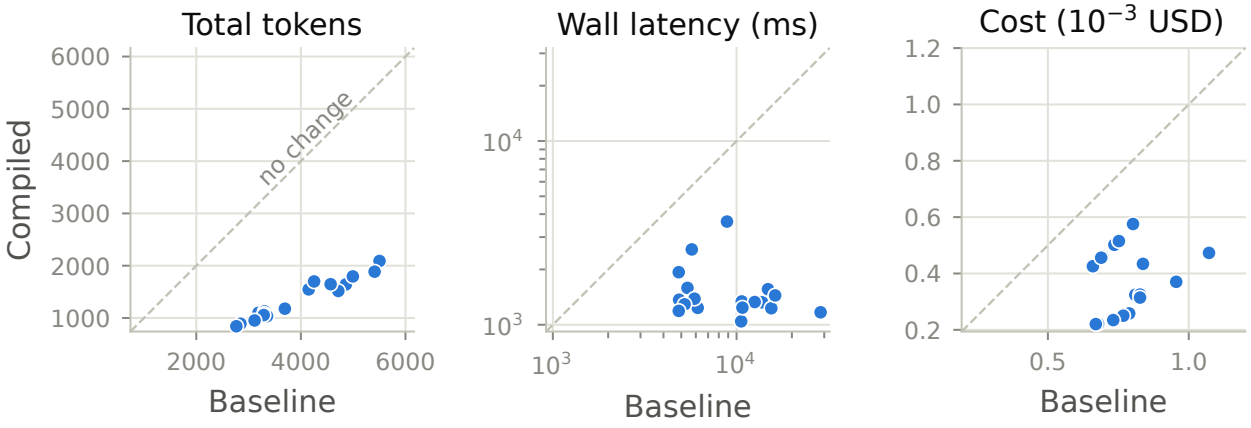

Figure 5: The same 18 pairs, unaggregated. Every issue is one point, baseline against compiled, so the reduction is visible as a distribution rather than a mean: no pair crosses the no-change diagonal on any of the three endpoints. Latency is plotted on log axes because the baseline spans an order of magnitude.

(fig. 5). All 18 pairs pass the registered contract, but zero discordances still permit a 15.3% one-sided 95% upper bound on degradation, so this is not an equivalence result. Discovery consumed 528 provider requests and 533,293 tokens; measured break-even ranges from 176 future episodes when amortizing requests to 292 when including confirmatory-arm cost, before engineering, review, monitoring, and cache effects. Table 14 gives the per-metric absolute means.

5.4 Harder shapes, partial compaction, and where the saving inverts

The stress suite confirms the intended boundary behavior (fig. 6). In Demo E, an order-fulfillment exception workflow branches over reads and pagination before a mandatory irreversible write. The live SDK run therefore compacts only the read prefix: model requests fall from 7.0 to 2.0 and total tokens by 66.4%, while all four baseline and compacted episodes pass the registered task contract (1.00/1.00). The measured estimated cost rises 8.3% because deleting turns fragments prompt-cache reuse. This is a fictional deterministic WMS fixture, not a real fulfillment system or business-record study.

The write remains under the ordinary agent, and the current Demo E program is a manually authored straight-line/branch fixture used to test the runtime effect boundary rather than evidence that automatic synthesis can safely infer arbitrary

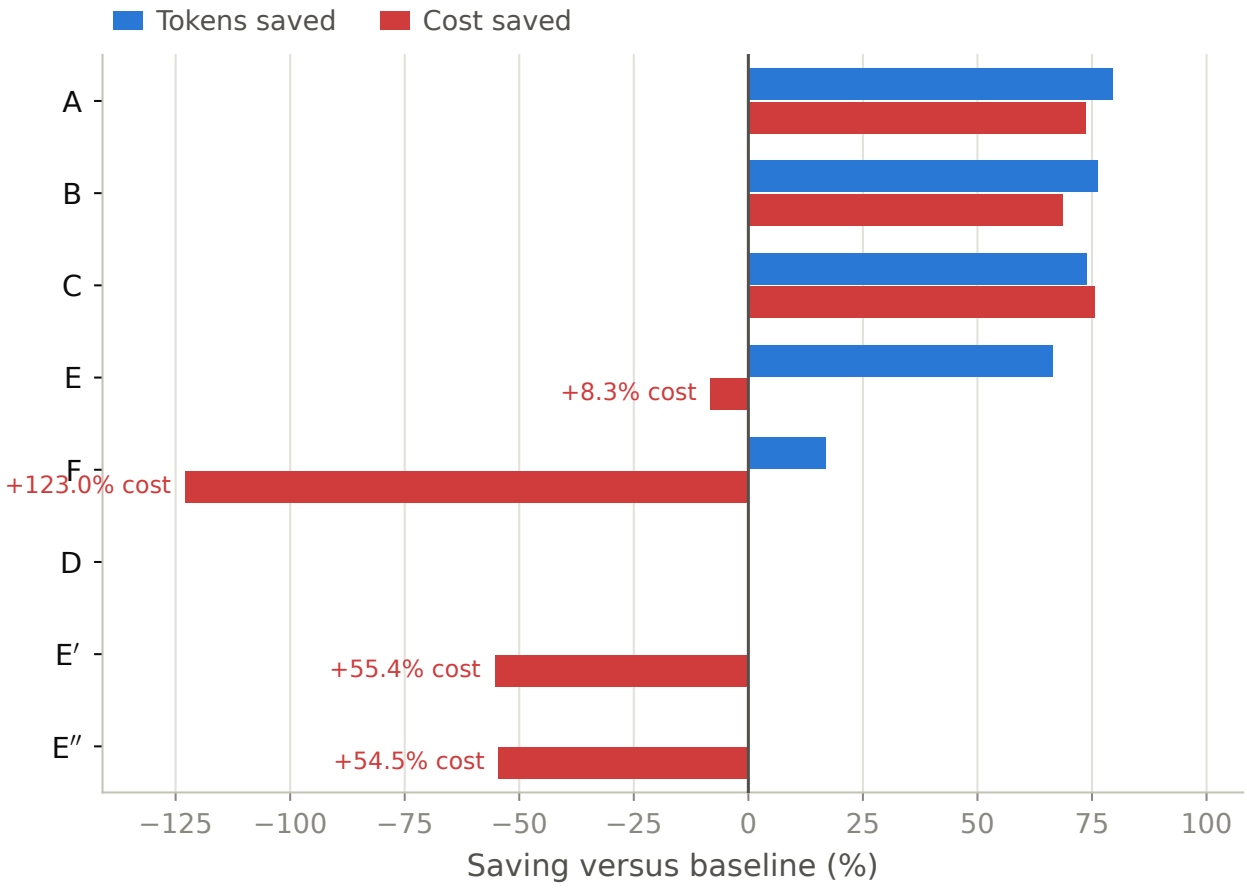

Figure 6: Controlled stress suite. Three of the eight conditions are negative controls whose only correct outcome is no compaction, and two of the compacting rows cost *more* than their baselines once prompt-cache reuse is fragmented. Table 16 names the workflow shape behind each condition and gives the per-demo counts.

writes. An undeclared MCP surface, an unsupported loop-bearing artifact, and a drifted entry schema all reproduce the baseline model-call count and quality. These refusals are the intended result: the paper uses the suite to establish partial-compaction and fail-closed behavior, not a production write-safety guarantee.

5.5 HMDA public-record preflight: a validated extension, not an optimization result

The HMDA checkpoint makes the proposed second domain concrete without overstating what has been run. The frozen pool contains 420 privacy-modified public LAR groups, and an independent validator reproduces all 420 exact gold records; 416/420 (99.05%) expose a variable path for a future trace comparison. The validator executed zero provider calls. Consequently, HMDA contributes a verified data, schema, privacy, and provenance boundary only. It contributes no measured baseline, GRC, or reviewed-macro quality, request, token, latency, cost, determinism, or workflow-reduction result, and it remains outside the primary effectiveness denominator. The SEC row in table 2 is retained as an explicit source-gated negative disposition rather than silently dropping an incomplete three-domain protocol.

5.6 The comparator that matters: a hand-written composite tool

For a workflow whose reads are already known, the obvious engineering answer is not a compiler: it is to write one composite tool that performs them (fig. 7). A request reduction measured only against an unoptimized agent cannot distinguish "compilation works" from "this workflow was easy". Table 8 therefore reports both against the same baseline on the offline stress study, and the result is not in our favour.

Table 8: GAC against the obvious engineering alternative — one hand-written composite tool performing the same reads — on the deterministic offline stress study. R_{req} is the paired request ratio against the unchanged agent, so lower is better and 1.000 means no saving. **The macro is at least as good on three of five workloads.** This substrate is simulated: a scripted policy stands in for the model at each boundary, so these numbers cannot settle the live question. They can, and do, settle whether the comparison was run.

Demo	Workload	Macro R_{req}	GAC R_{req}	Lower is better
A	Tier-1 support evidence gathering	0.726	0.755	**macro wins**
B	permissioned RAG knowledge assistant	0.335	0.718	**macro wins**
C	multi-agent incident triage	0.922	0.780	GAC wins
D	multi-tenant MCP operations (negative control)	0.724	1.000	**macro wins**
E	order-fulfillment exception handling	0.661	0.642	GAC wins

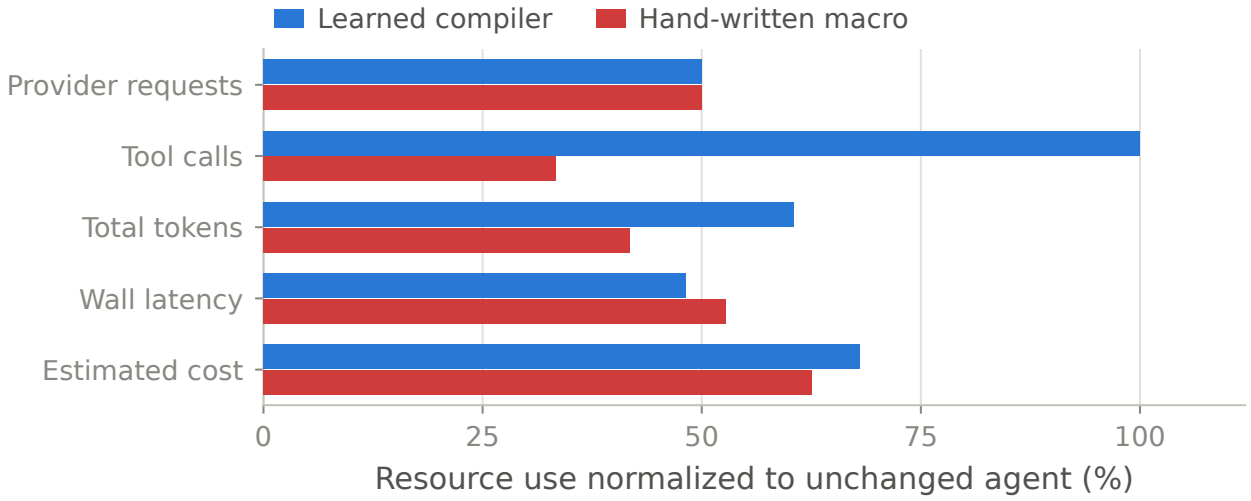

Figure 7: The live comparison that constrains the contribution, on the 30-pair expanded replication. The hand-written macro is at or below the compiled condition on every resource except observed wall latency, and uses one tool call where the compiler keeps three.

On the permissioned-retrieval workload the macro is far better (R_{req} = 0.335 versus 0.718): it collapses six reads into one call, whereas GAC must keep each call separately dispatchable to verify its live-outs. On Tier-1 support the macro is slightly ahead. GAC wins where the *choice* of which reads to make is itself part of the recurrent structure — multi-agent triage (0.780 versus 0.922), where a composite tool cannot remove the coordinator's routing turn, and fulfillment (0.642 versus 0.661), where branch-dependent evidence means a single fixed composite would over-read.

The negative control is the most informative row. A hand-written macro reaches R_{req} = 0.724 on the MCP workload; GAC reaches 1.000, because the tool effects are undeclared and the catalog fails closed. The macro takes a 27.6% saving that GAC refuses. Whether that refusal is prudence or lost value depends entirely on whether those undeclared tools really are read-only — which is exactly the question a human answers by writing the macro and the compiler declines to guess. Neither number is "right"; the pair is the point.

We therefore do not claim that GAC dominates ordinary engineering. The expanded live comparison in table 6 reaches the same conclusion directly: both candidates pass 30/30, but the macro matches the partial compiler's request count and uses fewer tools, tokens, and dollars; only observed wall latency favors the compiler. The earlier aggressive artifact saves one additional provider turn but records a factual miss. Where a hand-written composite is available and its effects are understood, write it. GAC earns its complexity only when the region is not economically maintainable as a manual macro, is branch-dependent, or requires guards a macro would otherwise assume implicitly.

5.6.1 Why the token win and the dollar win are different sizes.

The macro's advantage shrinks once prefix reuse is priced, and the reason was recorded all along. In the issue-type family the hand-written macro uses 30.9% fewer total tokens than the compiled condition but is only 8.0% cheaper,

Table 9: Prompt-cache structure of the primary families, recovered provider-free from the retained per-record usage of the same runs. "Input" is mean input tokens per episode, "Cached" the share of them served as cache reads, "Cache-write" the mean tokens written per episode. Cache reads are already priced by each study's frozen price table; the shares explain why token and dollar changes diverge.

Family	Condition	Input	Cached	Cache-write
Issue type	unchanged	4062	32.3%	1177
	GAC	2435	27.8%	1165
	hand-written	1633	0.0%	828
PR outcome	unchanged	2592	0.0%	185
	GAC	458	0.0%	0
	hand-written	458	0.0%	0
Backlog	unchanged	2689	0.0%	77
	GAC	454	0.0%	0
	hand-written	454	0.0%	0

Table 10: Fresh real-record comparison after adding guarded composite synthesis. "Exposed" counts model-visible interfaces; both conditions execute three source reads. GCS runs the admitted composite before the first provider request. This is a 12-pair exploratory extension designed after the earlier macro result, not a confirmatory replacement for table 6.

Condition	Requests	Exposed	Reads	Tokens	Wall (s)	Cost	Exact
Provider-visible macro	2.0	1.0	3.0	1524.0	2.49	$0.000430	12/12
Pre-model GCS	1.0	1.0	3.0	931.2	1.49	$0.000291	12/12

because it retains no cache reads at all (0.0% of its input, against 27.8% for the compiled condition and 32.3% for the unchanged agent; table 9). Collapsing three reads into one call shortens the prompt and simultaneously destroys the repeated prefix that made the remaining input cheap. A token-only objective scores that trade as a clear macro win; a cache-aware objective scores it as a near-tie. This is the effect the prompt-cache literature [24, 25] describes for compression, arising here from interface fusion.

The same table exposes a limit on our own cost numbers. The two newer families record zero cache reads in every arm, so their 75.1–75.3% cost reductions are measured against a cache-cold baseline, whereas the issue family's 32.0% is measured against a baseline that serves a third of its input from cache. Part of the width of the reported 32.0–75.3% range is therefore a property of cache warmth rather than of the compiled programs, and a production baseline with a warm prefix should be expected at the low end. We report the shares instead of narrowing the range post hoc, but a cache-controlled replication is the correct fix and we do not have one.

Amortization for the families that ship. Each family's compiler learned from 132 paid discovery episodes: $0.1146, $0.0931, and $0.0988 for issue-type, PR-outcome, and backlog routing respectively, at 528 provider requests each. Against the measured per-episode dollar saving, provider-side break-even is 411 episodes for issue-type routing and 182 and 181 for the two newer families — so the weak two-read artifact needs more than three times its own discovery cohort to repay itself in provider spend alone, before any engineering, review, monitoring, or invalidation cost. The 411/182/181 spread tracks compiled depth, not workload volume, which is the practical form of the depth sensitivity section 8 describes.

The result has a direct mechanistic explanation and a testable hypothesis. The macro is allowed to redesign the application interface: it fuses three reads, returns a task-specific sufficient record, and exposes one tool schema. The compiler is constrained to preserve existing interfaces, provenance, intermediate evidence, and fallback semantics; in the expanded run its groundability proof permits only a two-read prefix and all three logical reads remain necessary. We therefore hypothesize that manual macros dominate on stable, low-entropy workflows, while guarded compilation earns its cost across branching or changing workflow families for which manual discovery and maintenance do not amortize. This study establishes the structural facts behind that hypothesis, but it does not measure engineering effort, drift response, or the cross-workflow break-even.

Table 11: Fresh real-record evaluation after bounded prompt optimization. Values are means over six held-out issues. "Interfaces" counts model-visible tool calls. GEPA retained its seed; the combined row is therefore a GCS replication, not a distinct optimized prompt. Optimization overhead is excluded.

Condition	Requests	Interfaces	Input	Total	Wall (s)	Exact
Unchanged	4.0	3.0	3542.0	3683.5	5.08	6/6
GEPA (seed retained)	4.0	3.0	3530.0	3669.3	4.33	6/6
GCS	1.0	1.0	770.8	861.0	1.61	6/6
GCS + retained seed	1.0	1.0	770.8	868.8	1.78	6/6
Manual pre-model	1.0	1.0	770.8	865.5	1.46	6/6

5.7 Exploratory GCS: closing the measured macro gap

We implemented the interface transformation suggested by the comparator rather than discarding the guard. The compiler now canonicalizes only application-declared task-semantic arguments, emits the complete three-read program behind one projected interface, retains all internal provenance and verification, and executes it before the provider only when the continuation manifest matches exactly. Provider-free reconstruction of all 132 retained discovery traces dispatches 124, safely falls back on eight, and produces 124/124 exact projections with no projection failure. That replay makes no new model call and supports implementation correctness, not an efficiency claim.

This artifact was calibrated at $\alpha = .10$, not at the registered $\alpha = .05$ (table 1). Its gate admits 88 of 92 calibration groups with zero violations for a simultaneous upper bound of 0.0520, which exceeds .05; at the registered level the family would have retired and there would be no GCS result to report. Everything below is therefore a 10%-selective-risk result, and it is not interchangeable with the three primary families.

The paid comparison uses 12 further public issues excluded from all 424 issue identifiers used by the preceding experiments. Both the provider-visible hand-written macro and GCS pass all 12 exact factual and tool contracts; with no observed discordance, exact McNemar $p = 1$, and the one-sided 95% zero-event bound still permits a 22.1% population discordance rate. Relative to the measured macro, GCS reduces provider requests from 2.0 to 1.0 (−50.0%), total tokens from 1,524.0 to 931.2 (−38.9%), mean wall latency from 2.49 to 1.49 seconds (−40.0%), and estimated cost from \$0.000430 to \$0.000291 (−32.3%). Paired request, total-token, latency, and cost differences all have two-sided exact signed-rank $p = 0.000488$; output-token reduction alone is uncertain ($p = 0.155$). Each condition exposes one interface and still performs three source reads, so this is decision and context compaction, not elimination of necessary evidence.

The mechanism is placement, not magic: the provider-visible macro spends one request choosing its composite and another producing the answer, whereas GCS supplies the guarded projection to the first request. The follow-up in section 5.8 implements the previously missing manually engineered, equally guarded *pre-model* comparator. The run also covers only five bug and seven other-labelled issues—no question or enhancement cases survived strict prior-cohort exclusion. A retained two-case smoke run had worse GCS latency, reinforcing that the 12-case latency magnitude is provider- and schedule-sensitive. The supported conclusion is therefore precise: automatic guarded interface synthesis matches the measured macro's observed quality and beats its measured provider work on this family, while preserving safe fallback. It is not evidence of an advantage over manual placement.

5.8 Fair placement and bounded prompt optimization

All five conditions pass all six exact factual and tool contracts. The GCS arm executes the same $\alpha = .10$ artifact as section 5.7, so this comparison inherits that looser risk level (table 1). Relative to the unchanged agent, GCS reduces provider requests 75.0%, exposed interfaces 66.7%, input tokens 78.2%, total tokens 76.6%, observed wall time 68.2%, and estimated cost 67.9%; the paired structural and token reductions have exact signed-rank $p = .03125$. These six pairs remain too few for a population equivalence claim.

The fair manual program is the decisive comparator. It also uses one request, one exposed interface, and 770.8 mean input tokens, and passes 6/6. Against it, GCS changes none of those structural metrics. GCS emits 4.5 fewer output and total tokens on average ($p = .25$), while observed wall time and cost differences are also non-significant ($p = .844$ and .625). Automatic discovery, provenance, compatibility pins, and calibrated admission may justify GCS operationally, but this study shows no runtime dominance over a correctly placed hand-authored program. Manual construction and review effort were not measured.

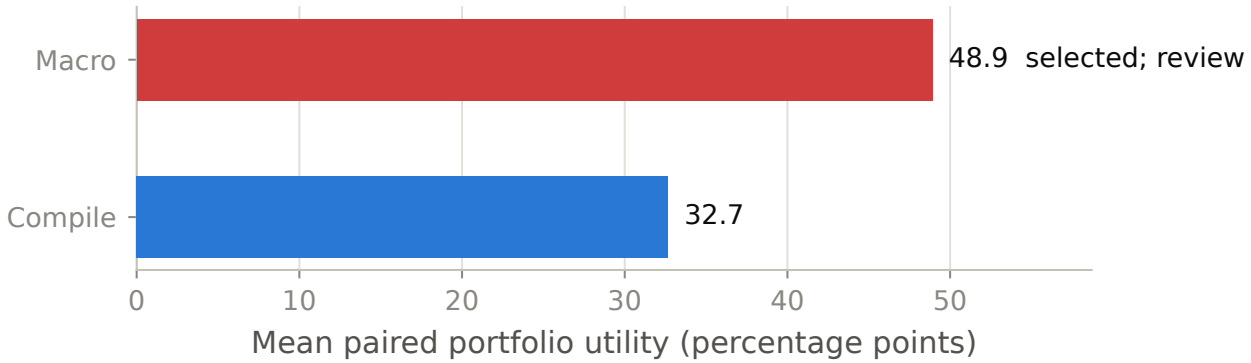

Figure 8: The frozen selection, taken over the 30 independent calibration issues. Both actions clear the risk limits, so the decision is made on mean paired utility alone and the measured macro wins it. Selection routes the action to human review; it does not deploy it. Table 15 gives the risk bounds and the fresh-cohort outcome.

GEPA consumes 14 real task evaluations and three real reflection calls under the fixed budget, proposes three alternatives, and retains the seed. Its held-out arm therefore keeps four requests and three interfaces; total-token reduction versus unchanged is 0.38% ($p = .688$), and estimated cost is 3.6% higher ($p = .688$). Optimization itself consumes 59 provider requests, 63,954 tokens, and an estimated $0.01163, reported outside the held-out arm. Because no prompt change is selected, the combined arm cannot support a composition claim. This is a bounded negative result on one extractive workflow, not a general failure of GEPA. One GCS+seed provider-span record exceeds its surrounding host wall time; the raw span is retained, but provider-span comparisons involving that arm are invalidated.

5.9 RQ5: prospective portfolio selection

Both actions record zero quality failures and zero non-positive-utility groups among 30 independent calibration issues. With the multiplicity split, each one-sided upper bound is 0.1359, below the pilot limit 0.15. Mean utility is 0.327 for compilation and 0.489 for the macro, so the frozen selector recommends the macro for review before seeing a fresh 12-issue cohort (fig. 8). Baseline and macro then pass 12/12 exact contracts; the macro halves provider requests, reduces tool calls from three to one, and lowers total tokens 59.2% and estimated cost 40.6%. This is a prospective action evaluation, not evidence that the selector beats an always-macro policy: calibration and test contain one workflow family, and the static policy would make the same choice.

5.10 RQ4: failed pilot and determinism

The first live pilot compiled suffix-only regions even though the runtime resolves only at entry. It therefore dispatched label/comment reads before their intended position, duplicated or reordered tools, and achieved only 16.7% task/tool-contract validity despite reducing requests (fig. 9). We archived the complete pilot, added the prefix invariant in eq. (2), added a regression test, and excluded every pilot issue from the final cohort. The corrected compiler blocks 116 suffix candidates and restores 100% observed contract validity.

On six repeated test issues, category decisions and tool traces agree perfectly in both conditions. Byte-exact natural-language answers agree in one of six baseline pairs and zero of six compiled pairs. Compaction therefore improves structural determinism of the tool prefix by construction, but not free-form surface determinism. This small sample does not support a claim that it stabilizes final text.

5.11 What did not work, and what each failure rules out

Five results in this section are negative, and they constrain the contribution more sharply than the efficiency numbers do. They are collected here because each one closes a different explanation that a reader would otherwise be entitled to keep open.

Every externally sourced recurrent family retired. Twelve synthesized NESTFUL families and both API-Bank families reached held-out replay without a single wrong execution — 24 passes and 12 abstentions on NESTFUL, two abstentions on API-Bank — and still none reached the 92 zero-violation groups the configured gate requires (section 5.1). This rules out the reading that recurrence plus successful replay is sufficient evidence to rewrite an agent, which is

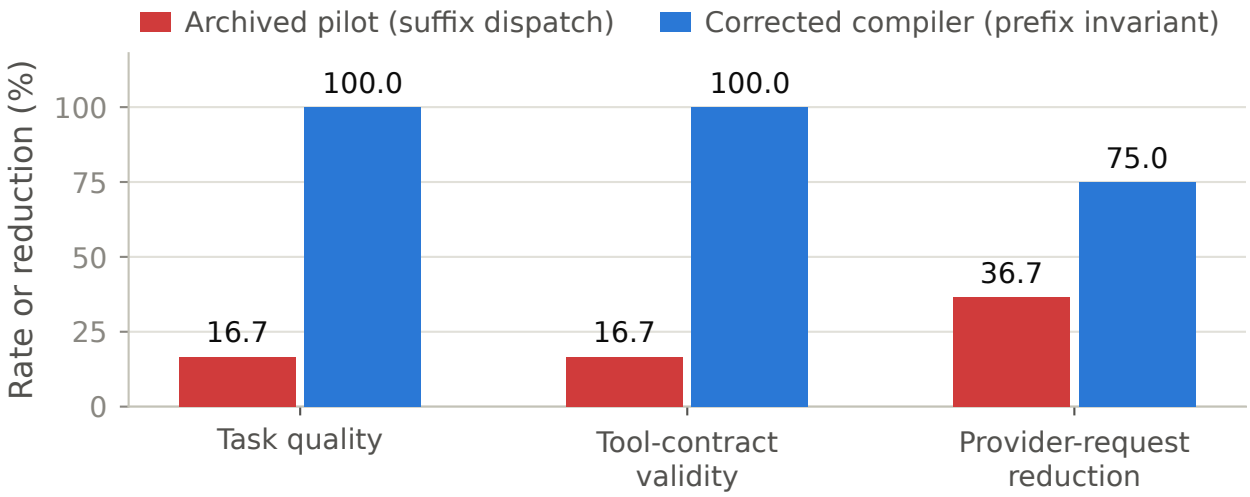

Figure 9: Why position is an invariant and not a tuning parameter. The archived pilot dispatched a compiled region from the wrong boundary and still reduced provider requests, which is exactly what makes it dangerous: efficiency survived while task quality and tool-contract validity collapsed to 16.7%. An optimizer scored on reduction alone would have accepted it.

the assumption a recurrence-only macro miner makes. It also fixes the cost of the guarantee: on these corpora the admission requirement, not the synthesizer, is the binding constraint.

The suffix pilot reduced requests while destroying the task. It cut provider requests 36.7% at 16.7% contract validity (section 5.10). This rules out efficiency-scored evaluation of this class of optimizer: a reduction metric ranked the broken compiler above no compiler at all.

The gate did not discriminate. Five of six live gates admit all or none, and the single partially selective gate refuses four of its 92 groups — which is precisely what lifts its own bound above the registered 5% (section 8). This rules out any claim that the calibrated score separates risky inputs from safe ones on this evidence; what the mechanism demonstrably provides is a sample-size requirement and a refusal, not a demonstrated risk–coverage frontier.

Bounded prompt optimization selected no change. GEPA spent 59 provider requests and retained its seed (section 5.7). This does not rule out prompt optimization in general — the budget was small and the workload narrow — but it does rule out the specific alternative explanation that the compiler's gains are available from prompt optimization alone on this task.

Hand-written programs matched the compiler. Manual programs reach 90/90 on both new families, and an independently authored pre-model program ties GCS at 6/6 with identical request and interface counts (section 5.6). This rules out runtime dominance as the contribution and leaves automatic discovery under a stated admission rule as the part of the claim the evidence still supports.

6 Related Work

Agent execution and scheduling. ReAct interleaves reasoning with environment actions [1]. LLMCompiler generates a dependency plan for the current request and schedules independent calls in parallel, reporting up to 3.7× latency and 6.7× cost improvement over ReAct [2]. GAC instead learns from repeated executions and deletes historical model boundaries; it currently executes synthesized calls sequentially. The methods are complementary.

Workflow and agent optimization. DSPy and MIPRO optimize instructions and demonstrations in declarative LM programs [3, 4]. RouteLLM selects a model based on preference data [6]; AgentSlimming prunes or replaces multi-agent graph nodes and reports token reductions up to 78.9% [5]. FlowCompile explores model, reasoning-budget, and workflow configurations for a declared graph and reports up to 6.4× speedup [26]. These optimize parameters or declared topology, whereas GAC infers a value-grounded replacement from observed traces.

Table 12: Adjacent optimizers by what they consume, what they rewrite, which safety conditions are hard rather than learned, and what licenses deployment. The last column is where GAC differs: every other row admits on a metric, a profile, or a test, none on a finite-sample selective bound.

System	Source	Rewrite target	Hard safety	Admission
LLMCompiler	Current-query scheduling	Parallel tool calls	No	No
DSPy / MIPRO	Prompt/program parameters	Metric optimization	No	No
GEPA	Execution + evaluation traces	Reflective prompt evolution	No	Held-out metric + Pareto
AgentSlimming	Multi-agent graph	Node pruning/quantization	No	Baseline rule
AWO	Trace tool sequences	Deterministic meta-tools	No	Empirical
Agent JIT	Task description	Validated code + schedule	Tool pre/post invariants	Candidate validation
EvoC2F	Plan IR + trajectories	DAG compiler + macro-skills	Effects + resources	Tests + contracts
FlowCompile	Declared workflow	Configuration Pareto set	No	Profiled
COVENANT	Declared policy	Workflow CFG	Effects	Runtime checks
GAC (ours)	Observed value flow	Decision-eliding program	Provenance + effects	Exact selective bound

GEPA uses execution and evaluation traces as natural-language feedback for reflective, instance-wise Pareto prompt evolution [27]. Across its six reported tasks it improves over GRPO with up to 35× fewer rollouts and produces instructions up to 9.2× shorter than MIPROv2 prompts. GEPA therefore prevents us from claiming trace-driven agent optimization itself as novel. It changes residual prompts rather than deleting model boundaries, making it complementary to region compilation. Our bounded factorial comparison covers unchanged, GEPA-only, GCS, combined, and manual pre-model conditions on six held-out records. Official GEPA 0.1.4 retains its seed, so the combined arm is a GCS replication rather than evidence of interaction. This budget-limited negative result does not contradict GEPA's broader task results.

AWO is the closest trace-based system: it identifies recurring tool-call sequences and bundles them into meta-tools, reducing LLM calls by up to 11.9% [8]. Agent Workflow Memory induces reusable workflows offline or online and reports large relative success gains on Mind2Web and WebArena [28]; it optimizes *what* an agent reuses rather than proving a reuse admissible, and it is the learned-workflow comparator a future head-to-head most needs. Agentic Compilation generates a deterministic JSON workflow from a single model call and replays it without further inference, reporting 80–94% zero-shot compilation success on web automation [29]; it shares our compile-versus-rerun motivation and differs in admitting the compiled plan without provenance, effect, or finite-sample risk conditions. A recent survey separates reusable templates, run-specific realized graphs, and traces [30]; in that taxonomy GAC optimizes traces under a declared-effect contract. EvoC2F compiles a typed Plan IR carrying dependency, effect, resource, idempotency, and retry annotations, then admits trajectory-derived macro-skills through tests, contracts, and regression checks [31]. Agent JIT Compilation generates and validates code plans for current web tasks, selects low-cost candidates, and schedules execution under tool pre/post-state invariants [32]. Both are closer compiler comparators than prompt-only optimizers. GAC differs by mining recurrent cross-execution regions from value-level traces and attaching a conditional group-level contract bound; the present experiments do not establish superiority to either system. Agentic plan caching adapts reusable plan templates and reports 46.62% mean cost reduction [9]. GAC adds explicit typed provenance, effect and position barriers, hard compatibility contracts, verifier/staging semantics, and an exact selective admission rule. Our result should not be read as a head-to-head improvement: we did not reimplement AWO on the GitHub scenario.

COVENANT compiles natural-language policies into workflow graphs and checks proposed actions against the declared procedure [33]. It uses policy as source; GAC uses executions as candidate evidence and must therefore abstain more aggressively. Combining declared workflow authority with trace-derived specialization is promising.

Tool-use evaluation. BFCL covers serial, parallel, abstaining, and stateful function calling [34]; API-Bank provides runnable APIs and tool-use dialogues [35]; ToolSandbox evaluates stateful conversational interactions [36]; τ^2-Bench adds dual control by agent and user [37]; ToolBench scales tool retrieval [38]; AgentBench spans interactive environments [39]; GAIA and BrowseComp test general assistant and persistent browsing capabilities [40, 41]; and SWE-bench grounds agents in real software issues [42]. These primarily test whether an agent can choose and execute actions or

produce a final answer. NESTFUL exposes nested producer references, while API-Bank retains recorded call results, making those two suitable for our post-trace question [18]. The artifact retains a revision-pinned supplementary interoperability ledger for the other benchmarks; we do not turn absent trajectories into compiler failures. Stateful and write-bearing compilation remains future work until transactional staging exists.

Trace-based specialization, guards, and deoptimization. The architecture is, structurally, a tracing JIT for agent executions, and the correspondence is close enough that the vocabulary is nearly shared: hot-trace detection becomes family mining, guard synthesis becomes the hard guard, region compilation becomes bounded synthesis, and deoptimization to the interpreter becomes fallback to the baseline agent. Dynamo introduced transparent trace regions with guards and fallback [43], and trace-based type specialization [44] and meta-tracing [45] developed the guard-and-recompile discipline we borrow. The problem of section 3.7.1 — restoring a consistent state after a specialized region is abandoned — is exactly dynamic deoptimization [46], and that literature's answer, explicit deoptimization points, is the same mechanism we defer to as "control points with continuation tokens". Effect systems supply the discipline behind treating effect declarations as a trusted computing base [47], and selective prediction long predates learn-then-test: the reject option is Chow's [48]. What is not inherited is what an agent guard must additionally certify: that arguments are grounded in observable state, that declared effects permit speculation, and that the residual error rate carries a finite-sample bound. We take the framing as the clearer statement of the contribution.

Behavioral equivalence and caching economics. Counterfactual trace auditing pairs with-skill and without-skill traces, aligns their phases, and shows that unchanged aggregate pass rates can hide many substantive behavioral differences [49]. That is exactly the weakness of our registered-contract oracle, and it describes the instrument a stronger equivalence check would use. On economics, prompt-cache strategy has been evaluated across three providers on long-horizon agentic tasks, reporting 41–80% API-cost reductions and showing that naive full-context caching can even increase latency [24]; cache-aware prompt compression models the compression/caching crossover directly [25]. Our section 5.4 observation that fewer tokens can cost more is therefore *consistent with* that literature rather than a new discovery. What our data adds is the same effect arising from *turn removal and route fragmentation* rather than from compression, and the consequence that a workflow optimizer's objective must price prefix reuse.

The deployed counterpart of that literature is context-compression middleware. Headroom, for example, sits between agent and provider and compresses tool outputs, logs, files, and retrieved chunks, reporting 60–95% token reduction on JSON payloads, 15–20% on coding tasks, and 73% on a GitHub triage workload, with a `CacheAligner` whose stated job is to flag content that would bust a provider KV-cache prefix [50]. The comparison is worth stating precisely because the token numbers are superficially similar to ours on a superficially similar workload. Such systems shrink the payload of calls that still happen; they never remove a model boundary, so they cannot reduce provider requests at all. GAC removes the boundary and leaves the payload alone. The two therefore compose rather than compete — one could compress the reads that a compiled prefix still performs — and the honest consequence is that our token reduction has a comparator we did not run. A compressed baseline is the single most informative addition to the resource story in this paper, and section 8 records it as unmeasured. Their accuracy evidence is also of a different kind: measured benchmark deltas over lossy compression, with no abstention path and no finite-sample bound, which is the same distinction that separates GAC from every other row of table 12.

Program analysis and selective risk. Canonical trace families resemble process-mining variants [51]; bounded synthesis relates to partial evaluation [52], likely-invariant mining [14], and programming by example [12]. Risk-controlled admission builds on exact binomial inference [15] and learn-then-test calibration [16]. The guarantee remains conditional on i.i.d. or conditionally i.i.d. admitted group indicators and a frozen candidate; provenance and effects are separate hard constraints, not learned away by calibration.

7 Discussion

7.1 What is novel, and what is not

Core novelty. The contribution is not agent compilation; it is *admissibility for trace-derived specialization*. Deterministic workflows [29], reusable trace patterns [8, 28], and effect-aware plans all have clear precedent. A classical JIT guard checks types and shapes. An agent guard must additionally show that arguments are grounded in observable state, declared effects permit speculation, runtime position is safe, and observed contract risk is bounded. GAC combines those requirements in one compile-or-retire path: typed value provenance, hard effect/permission/compatibility barriers, bounded readable synthesis, verifier and staging semantics, and per-candidate exact finite-sample admission. Each element has predecessors; their composition defines the new safety argument. The statistical component remains the

weakest empirically because the observed gate is all-or-none rather than a demonstrated risk–coverage discriminator (section 8).

GCS adds an interface-level specialization over this guarded program: a bounded projection of verified live-outs, a signed task-semantic argument contract, and an exact continuation pin allow the region to execute before the first provider request. The individual ideas are not new—manual macros, projection pushdown, and partial evaluation are standard—but their composition preserves the compiler's internal effect checks, provenance, staging, and fallback rather than treating a generated meta-tool as an opaque trusted shortcut. The 12-pair result supports that mechanism on one family. The later six-pair comparison finds structural parity with an independently authored pre-model program, so the scientific claim is automatic guarded specialization, not superior runtime efficiency once a human has already placed the same program correctly.

The portfolio adds a second, narrower composition: selection across transformation *classes* rather than configurations within one declared graph, with separate exact bounds on task failure and non-positive paired utility, compatibility invalidation, and a human-review condition. FlowCompile already constructs reusable accuracy–latency configuration sets and downstream routing; GEPA evolves residual prompts; AWO and Agent JIT generate reusable or task-specific execution structures. We therefore do not claim that portfolio search itself is new. The distinct object here is an evidence-gated choice among baseline, compiler, and reviewed application-interface redesign. Its current evidence is only one family and does not demonstrate that this choice rule beats a fixed macro policy.

The NESTFUL result is as important as the live saving. A system that emitted all 32 recurrent families would look more productive, yet none has the 92 zero-violation group records required by the configured calculation. Treating those records as independent is a load-bearing sampling assumption, not a property established by the benchmark. Refusal exposes the data requirement rather than moving it into an undocumented heuristic.

7.2 What the experiments changed about the design

Seven findings altered the method after it was first built, and each is stated here as the observation that forced the change rather than as advice.

Per-episode saving does not determine whether compilation pays. Provider-side break-even is 411, 182, and 181 future episodes for the issue-type, PR-outcome, and backlog-attention families, and these exclude engineering, review, monitoring, and invalidation cost entirely. The issue-type figure is the largest because its admitted program is only two reads deep, so the family with the safest artifact is also the slowest to repay it. Amortization, not reduction, is the quantity that separates a workflow worth compiling from one that is not.

Token reduction and cost reduction came apart. The hand-written macro uses 30.9% fewer tokens than the compiled arm but is only 8.0% cheaper, because it reads 0.0% of its input from prompt cache against the compiled arm's 27.8%. Deleting turns changes the number and shape of cache reads and writes, so the two quantities measure different things and a study reporting only tokens would have overstated the saving.

No amount of calibration data substitutes for an effect declaration. Demo D exposes undeclared MCP effects and is correctly refused; the refusal comes from the declaration, not from any statistical check, and the run is indistinguishable from baseline (3.0 model calls in both arms, tokens +0.1%). A misdeclared external write is therefore outside what the admission bound can protect, which places the effect catalog in the trusted computing base rather than in the learned part of the system.

Compiler and runtime must share control-point semantics, and the failure is silent otherwise. The archived pilot emitted suffix-only regions against a runtime that resolves at entry. It still reduced provider requests 36.7% while task quality and tool-contract validity fell to 16.7% (fig. 9) — an efficiency-scored optimizer would have accepted it. Making position an explicit invariant blocked 116 suffix candidates and restored 100% observed contract validity.

Interface and execution position are separable optimizations. Exposing a macro leaves a model turn to select it; a continuation-pinned pre-model composite removes that turn. GCS reduces provider requests 50.0% against the measured provider-visible macro while both pass 12/12, which isolates position as the operative variable rather than interface width.

Learned-optimizer overhead needs a separate ledger, because it can be unbounded by the benefit. Bounded GEPA consumes 59 provider requests, 63,954 tokens, and an estimated $0.01163 across 14 task evaluations and three reflections, then retains its seed. With no change selected there is no benefit against which to amortize that cost, so folding optimization spend into the held-out arm would have hidden a null result.

Opaque identifiers behave as nominal values, not measurements. The archived pull-request pilot projected nullable merge timestamps and hulled empirical numeric ranges over identifiers, which is correct for bounded quantities and

wrong for IDs: it rejects valid unseen values while admitting nothing safer. Retaining type and provenance with an `any` hull keeps the guard sound without that false rejection.

7.3 Open questions

Two of these are forced by section 5.4 rather than chosen. First, eq. (5) counts tokens and calls, and Demo F shows that is the wrong objective: a route whose traffic share is too small to amortize its own cache write is a net loss at +123.0% cost even though its prompt is strictly shorter. What the objective should charge for *cache-prefix fragmentation*, and whether that term can be estimated before synthesis rather than measured after it, is unresolved. Second, the break-even in section 5.3 is computed against the list input price, which overstates headroom for any workload whose baseline is already cache-dominated; the right comparison is the *cached* baseline price, and we do not know how much of the reported saving survives it.

Portfolio optimization beyond the pilot. The implemented portfolio layer selects among measured actions and can recommend a macro for review; it does not synthesize arbitrary macro code, execute cache-only or model-routing candidates, estimate developer effort, or learn a policy across workflow families. A study that could decide the selection question needs families deliberately spanning stable bundles, branching prefixes, cache-dominated routes, unsafe effects, and low-support abstention. It would have to measure construction, review, maintenance, invalidation, and drift costs and compare against always-macro, always-compile, cache-only, FlowCompile-style configuration selection, and a learned contextual policy. Only then can the central selection claim be stronger than "the pilot correctly chose the already stronger macro."

One further question is structural rather than empirical. Section 8 shows that the per-candidate bound does not compose: the compiler searches candidate families, but the guarantee is stated for one fixed candidate, and a two-candidate correction already moves the requirement from 92 to 106 zero-violation groups. Whether familywise risk can be allocated across the artifacts a compiler actually produces without making the data requirement prohibitive is the open problem that most limits what this method can claim, and it is not answered here.

8 Limitations and Threats to Validity

The threats below are grouped by the kind of inference each one endangers, so that a reader can tell which results a given caveat actually touches. Each is stated at the strength the evidence forces, including the four that undercut a headline number.

8.1 Internal validity

Whether the reported effect follows from what was manipulated rather than from how the runs were arranged or configured.

The original latency result is confounded by condition order. The fixed-prefix harness runs the entire baseline batch and then the entire compiled batch, without randomizing or counterbalancing order within pairs. Provider load, connection reuse, and prompt-cache warmth therefore differ systematically between conditions. The request-count reduction is structural and unaffected, but the -85.0% wall-latency figure should be read as an observation under this ordering rather than a clean causal estimate; the same confound is the most likely explanation for the refusing conditions of section 5.4 billing more than their baselines on identical token counts. Both natural-order studies counterbalance all six three-condition permutations; the expanded run assigns each exactly five primary records. This removes ordinal imbalance but cannot eliminate provider noise or cache interference.

Removing model turns can remove guardrail evaluations. Moderation, policy, and guardrail checks in agent frameworks commonly run *at* model boundaries. A region that deletes three of four boundaries therefore risks deleting three of four guardrail evaluations, and neither our effect catalog nor the read-only policy covers that: a guardrail is not a tool call, so it is invisible to both. Our demonstrations expose no guardrails, so the gap is untested rather than benign. A deployment must either re-run guardrails inside the permission facade for every elided boundary, or declare guardrail presence an explicit barrier condition alongside handoffs and approvals. Only the second is available here, because the first is neither implemented nor tested; a misdeclared external write remains undetectable by statistics — detecting it needs differential execution in a sandbox or declaration-versus-observation diffing, neither of which we implement.

Registry integrity is configured off in the reported run. Artifact records are mutable Python objects, the reported experiment sets lifecycle and approval fields directly, and the archived artifact carries an empty signature because

signing was not enabled. The immutability and signature-verification semantics described in section 3 are therefore capabilities of the registry, not properties this experiment demonstrates.

No compressed baseline. Every resource number in this paper compares a compiled prefix against an *uncompressed* baseline agent. Deployed context-compression middleware reports token reductions of the same order on a comparable GitHub triage workload by an entirely different mechanism [50], so our token, cost, and latency reductions should not be read as the best available on those axes — only as the reduction attributable to removing model boundaries. The provider-request reduction is not exposed to this threat, because compression never removes a call. Running GAC against a compressed baseline, and against the composition of both, is the most informative missing experiment in the resource story.

Baseline scope. The natural study runs both the unchanged SDK loop and a live hand-written composite tool. In the expanded partial-compilation study, the macro and compiler each use two provider requests and pass 30/30, but the macro uses one local tool call rather than three and is better on tokens and estimated cost. In the earlier aggressive study, GAC saves one more provider request and more tokens but records a factual miss. The offline suite independently gives the macro the lower request ratio on three of five workloads. The compiler itself remains compile-or-retire; the new portfolio layer selects the measured macro and emits a review-required recommendation. Separately, GCS packages an admitted read program behind a synthesized projection and beats the measured provider-visible macro on 12 exploratory pairs; it does not synthesize arbitrary application logic. The six-pair follow-up gives an independently authored program the same pre-model position and finds equal requests, interfaces, input tokens, and exact quality. Official GEPA retains its seed under a 14-task-evaluation, three-reflection budget. The portfolio does not generate the macro, and on a single family its decision remains indistinguishable from an always-macro rule. AWO, AWM, Agent JIT, EvoC2F, LLMCompiler, FlowCompile, and plan caching remain unexecuted. The evidence establishes an intervention and engineering trade-off, not state-of-the-art superiority.

8.2 Construct validity

Whether the measures capture the things they are named after.

Factual quality is narrow and depth-sensitive. The original five-boolean oracle does not check summary factuality and accepts fluent fabrication. The natural studies replace it with exact snapshot fields and source-supported excerpts scored independently of tool order, but this remains an automatic extractive contract rather than human semantic adjudication. The expanded two-read artifact passes 30/30 while the earlier three-read artifact catches one compiled-only Markdown-link alteration. Thus preservation is observed at one depth and explicitly unsupported at the other. Both audited oracle corrections show that even strict-looking metrics can drift; raw online labels and correction records are retained. In the earlier run the corrected eligibility rule would swap one discovery trace, but the evaluated artifact remains the one actually executed. In the expanded run the semantic regrade changes only literal tool-contract labels, not compiler inputs, provider outputs, or measurements. The post-hoc continuation replay repairs the retained exact-source miss with a task-specific deterministic renderer, but it is not a live evaluation and does not replace human semantic adjudication or a separately calibrated continuation-risk frontier.

Semantic scope. Recorded output equality and empirical contracts do not prove full semantic equivalence. The closed DSL intentionally misses legitimate transformations and loops. Tool-effect truth, permissions, freshness, and isolation are application-supplied. The current version is read-only, does not parallelize synthesized calls, and task-semantic argument contracts are application-supplied trusted declarations. The narrow SDK adapter bypasses streaming, hosted tools, MCP, handoffs, and loops.

8.3 Statistical conclusion validity

Whether the inferences drawn from the numbers are licensed by the design that produced them.

The selective gate did not discriminate in this run. On the sealed artifact, every grid threshold below 0.14 admits zero calibration groups and every threshold at or above 0.14 admits all 92, with zero violations throughout. The score therefore behaved as an all-or-none support test rather than as a ranking of safe against unsafe inputs, and six of its seven feature weights are exactly zero (only `hull_margin` is non-zero).

The cause is diagnosable rather than mysterious, and it is a property of the data, not of the estimator. The score is fitted on development-group *unproductive* outcomes, and the artifact passed all eight development replays — so the logistic

model was fitted with *zero* positive examples on eight observations over seven features. A single-class fit yields a near-constant score, and a near-constant score is exactly what produces the observed step at $\eta = 0.14$: all 92 calibration entries land in one interval. Reporting the positive count is therefore mandatory for any such gate, and we report it here as zero. Two honest routes exist: build a development set containing genuine unproductive outcomes and publish a risk–coverage curve at several α, or replace the learned score with an explicitly one-class construction (distance-to-hull, or a conformal nonconformity measure over entry features) and describe it as such. Until one of those is done, contribution (v) is demonstrated only as a sample-size counter: on NESTFUL the gate refuses everything because $26 < 92$, and here it admits everything because the score is constant. We cannot claim a demonstrated risk–coverage frontier from this evidence. Establishing one needs calibration and test sets containing natural error: hard-but-supported inputs, out-of-domain entries, source and schema drift, stale versions, ambiguous provenance, and partial tool failures.

Both natural-order artifacts repeat the same pattern. In the earlier run, thresholds below 0.14 admit no calibration records and those at or above 0.14 admit all 45, with zero registered violations and upper bound 0.0992 at $\alpha = .10$. In the expanded replication the step moves to 0.11, below which none of 92 records are admitted and at or above which all are, with zero registered violations and upper bound 0.0498. Two fitted weights are non-zero there, but they are numerically identical to fourteen significant figures (0.80093206584596 for both `hull_margin` and `provenance_ambiguity`), which is the signature of two perfectly collinear features in a degenerate fit rather than of a score that has learned to weigh them. The observed admission decision is still all-or-none. Natural planning and more calibration data therefore do not repair the missing risk–coverage discrimination, and the count of non-zero weights is not a useful progress measure for it.

One artifact is a partial exception, and it is the one whose bound is loosest. The guarded-composite gate of section 5.7 admits 88 of 92 calibration groups at its selected threshold, so its admission decision is not all-or-none: four groups are refused on entry-observable evidence. That is the only observed instance in this paper of the gate doing the job the method claims for it, it is a coverage of 0.957 rather than a curve, and it is precisely why that artifact's bound is 0.0520 instead of 0.0498 (section 3.5.1). A mechanism visible only at $n = 4$ refusals under a looser α does not establish a risk–coverage frontier; it does show that the degeneracy is a property of these development sets rather than of the estimator.

Statistical scope. The pilot and final cohort are separated, but the study was not externally preregistered. Secondary tests are unadjusted and provider latency is noisy. The optimizer comparison contains only six pairs; its exact signed-rank minimum is therefore .03125, and one provider-span record fails the host-wall consistency check. Clopper–Pearson is exact but conservative. Three selective-risk levels appear in this paper and are not interchangeable: $\alpha = .05$ for the three primary families and the prescribed-prefix ablation, $\alpha = .10$ for the earlier three-read artifact and for the guarded-composite artifact behind every GCS and comparator number, and a 15% pilot limit for the portfolio (table 1). The pooled 3.3% preservation bound in section 5.2 spans three separately calibrated artifacts and so is a statement about the observed 90 pairs, not a per-artifact certificate. The gate's guarantee applies to the recorded group-level violation definition only when admitted group indicators are i.i.d. or conditionally i.i.d.; drift, clustered tenants, adaptive artifact selection, or incomplete outcomes invalidate a naive interpretation. Each artifact is calibrated individually; the implementation lacks both candidate-family multiplicity control within a compile and a global guarantee when many artifacts are selected.

8.4 External validity and replication

How far the results carry beyond the corpus, model, and snapshot that produced them, and what a repetition would and would not recover.

The natural task still constrains the evidence shape. The original conformance prompt names three tools in an exact order, so its recurrence is constructed. The natural follow-ups remove tool names and order, and compiler eligibility no longer requires an exact trace, but the requested title, state, labels, and evidence excerpt still make the three sources useful. All 132 expanded discovery agents (and all 80 in the earlier study) converge on one order. This is stronger evidence of naturally selected recurrence than the original study, but it does not establish discovery in open-ended plans where tools may be skipped, substituted, branched, or revisited for reasons not fixed by an output schema.

Empirical scope. The richer workflow-family studies remain one repository/domain and one model configuration: the confirmatory compiler study contains 30 primary issues, the prospective portfolio pilot contains 12, the exploratory GCS comparison contains 12 further pairs, and the learned/manual comparator contains six five-condition blocks. A separate PR-outcome-core extension adds five repositories, 580 exact discovery traces, 120 completed time-forward held-out pairs, and one compile-time retirement, while a balanced rerun adds three repositories, 360 exact discovery traces, and 180 completed time-forward held-out pairs with full `open/merged/closed_unmerged` coverage under the

same task. Both extensions still use a narrower two-read task than the main workflow-family evidence. The GCS cohort contains only bug and other-labelled issues; the comparator contains bug, enhancement, and other but no question-labelled issue. All GitHub evidence uses a frozen public snapshot, not the live GitHub API, so network/service behavior, concurrent mutation, rate limits, and authentication are absent. The exact gates have 92, 45, and again 92 calibration records depending on the artifact, while the portfolio has only 30 selection groups and uses a looser 15% risk limit. The fresh portfolio test contains no question-labelled issue after strict prior-record exclusion. There is no human study of productivity or user experience and no multi-domain production canary.

The artifact includes a prospective three-domain extension harness for vulnerability evidence, SEC filing facts, and privacy-modified public HMDA records. The HMDA checkpoint is now reported explicitly: 420/420 independent exact-gold reconstructions pass and 416/420 groups expose a variable path, but zero provider calls have run. The SEC pool is absent because the required compliant source contact is not configured; consequently the three-domain protocol is not frozen and human macro approvals are absent. These artifacts demonstrate data-pipeline and control-plane feasibility only and remain excluded from every effectiveness result in this paper.

Portfolio scope. The weighted utility is a declared operator preference, not a universal scientific metric. Its current dimensions omit construction time, review effort, maintenance, drift, cache fragmentation, and authorization changes. Both measured actions pass both binary gates, so exact risk controls admission but does not explain the choice; the utility weights do. The prospective result validates the chosen arm against baseline, not the selector against alternative selection algorithms. A claim that the portfolio advances the state of the art requires multi-family decisions, closer learned baselines, sensitivity to the weights, and time-forward evaluation under drift.

Reproducibility scope. The original experimental workspace did not retain prior `.git` history. The public artifact begins from a later initial snapshot, so pre-snapshot commit ancestry and CI state cannot be reconstructed. Dataset revisions and file hashes are pinned, raw results are retained, and the full local suite passes on Python 3.14.4; nevertheless, provider responses and latency will not be byte-identical on rerun. API access and the named model are required to repeat the live study.

9 Conclusion

Guarded agentic compaction treats repeated traces as candidate evidence, not permission to rewrite an agent. It compiles only value-grounded, read-only prefixes that satisfy explicit compatibility, verification, and finite-sample admission requirements; otherwise it keeps the original agent.

The experiments show why each boundary matters. Across three real public GitHub workflow families, compiled programs reach 90/90 exact held-out contracts versus 89/90 for unchanged agents while reducing requests 50.0–75.0%, tokens 39.5–81.4%, observed latency 51.7–73.0%, and estimated cost 32.0–75.3%. A separate five-repository, time-forward PR-outcome-core extension reaches 580/580 exact discovery traces, completes 120/120 exact held-out pairs on four repositories, and retires the fifth at compile time; a balanced three-repository rerun then reaches 360/360 exact discovery traces and 180/180 exact held-out pairs while compacting `open`, `merged`, and `closed_unmerged` cases on all three. Across that balanced rerun, compiled execution reduces requests 66.7%, total tokens 78.4%, observed latency 60.7%, and estimated cost 72.7%, while the fixed two-read template remains essentially tied on the simplified task. A historical aggressive artifact still records a factual miss, and fairly placed hand-written programs also reach 90/90. What the experiments separate, then, is discovery and admission from runtime efficiency: the first two can be automated at a stated risk level, while the third is not attributable to compilation once a human has placed the same program correctly. NESTFUL and API-Bank add a different result: despite recurrent, executable traces, every family retires because the configured evidence requirement is unmet.

That separation suggests an evaluation criterion for this class of system: measure a workflow optimizer by the evidence it requires, the effects and positions it refuses, and the state to which it falls back—not only by the turns it removes. The criterion is proposed, not validated; applying it to systems other than this one is open. Deciding between the remaining candidate explanations would require a richer time-forward, cross-repository comparison on full workflow tasks, a non-degenerate risk–coverage frontier, measured manual construction and maintenance cost, cache effects, drift, and closer executable workflow-learning baselines.

Code Availability

Code, data manifests, and research artifacts are available at `https://github.com/rrahimi-uci/guarded-agentic-compaction`. What was studied is guarded read-only specialization: no experiment here

compiled a write, bypassed an approval, or touched a regulated decision, so the results say nothing about those cases in either direction.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[2] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 24370–24391, 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

[3] 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*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

[4] Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024. URL https://arxiv.org/abs/2406.11695.

[5] Yulang Chen, Haoxuan Peng, Jinyan Liu, Zichen Wen, Dongrui Liu, and Linfeng Zhang. AgentSlimming: Towards efficient and cost-aware multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 30064–30086, 2026. doi: 10.18653/v1/2026.acl-long.1387. URL https://aclanthology.org/2026.acl-long.1387/.

[6] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

[7] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://aclanthology.org/2023.emnlp-main.825/.

[8] Sami Abuzakuk, Anne-Marie Kermarrec, Rishi Sharma, Rasmus Moorits Veski, and Martijn de Vos. Optimizing agentic workflows using meta-tools. *arXiv preprint arXiv:2601.22037*, 2026. URL https://arxiv.org/abs/2601.22037.

[9] Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Cost-efficient serving of LLM agents via test-time plan caching. *arXiv preprint arXiv:2506.14852*, 2025. URL https://arxiv.org/abs/2506.14852.

[10] OpenAI. Agents SDK guide. OpenAI Developer Documentation, 2026. URL https://developers.openai.com/api/docs/guides/agents. Accessed 2026-08-03.

[11] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009. doi: 10.1561/1900000006.

[12] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 317–330, 2011. doi: 10.1145/1926385.1926423.

[13] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 404–415, 2006. doi: 10.1145/1168857.1168907.

[14] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, 2001. doi: 10.1109/32.908957.

[15] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404.

[16] Anastasios N. Angelopoulos, Stephen Bates, Emmanuel J. Candès, Michael I. Jordan, and Lihua Lei. Learn then test: Calibrating predictive algorithms to achieve risk control. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TNcOrox3tQ.

[17] Stephen Bates, Anastasios Angelopoulos, Lihua Lei, Jitendra Malik, and Michael I. Jordan. Distribution-free, risk-controlling prediction sets. *Journal of the ACM*, 68(6), 2021. doi: 10.1145/3478535. URL https://arxiv.org/abs/2101.02703.

[18] Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33538–33547, 2025. doi: 10.18653/v1/2025.emnlp-main.1702. URL https://aclanthology.org/2025.emnlp-main.1702/.

[19] Hugging Face. Github issues dataset card. Hugging Face Datasets, 2025. URL https://huggingface.co/datasets/helmo/github-issues. Revision e344be7b84d199661a9956036991e1fc25715a47.

[20] OpenAI. API pricing. OpenAI Developer Documentation, 2026. URL https://developers.openai.com/api/docs/pricing. Accessed 2026-08-03.

[21] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, 1993. doi: 10.1201/9780429246593.

[22] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945. doi: 10.2307/3001968.

[23] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157, 1947. doi: 10.1007/BF02295996.

[24] Elias Lumer, Faheem Nizar, Akshaya Jangiti, Kevin Frank, Anmol Gulati, Mandar Phadate, and Vamse Kumar Subbiah. Don't break the cache: An evaluation of prompt caching for long-horizon agentic tasks, 2026. URL https://arxiv.org/abs/2601.06007.

[25] Yan Song. Cache-aware prompt compression: A two-tier cost model for LLM API caching, 2026. URL https://arxiv.org/abs/2607.15516.

[26] Junyan Li, Zhang-Wei Hong, Maohao Shen, Yang Zhang, and Chuang Gan. FlowCompile: An optimizing compiler for structured LLM workflows. *arXiv preprint arXiv:2605.13647*, 2026. URL https://arxiv.org/abs/2605.13647.

[27] 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. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.19457. Oral presentation.

[28] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 63897–63911, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

[29] Jagadeesh Chundru. Agentic compilation: Mitigating the LLM rerun crisis for minimized-inference-cost web automation, 2026. URL https://arxiv.org/abs/2604.09718.

[30] Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for LLM agents, 2026. URL https://arxiv.org/abs/2603.22386.

[31] Lei Wei, Qi Liu, Ruiyang Huang, Xiao Peng, Sicong Xie, Lanbo Lin, Chenhao Jiang, Yuanwu Xu, Tianyuan Yang, Jiayao Liu, Li Cai, Zhaolu Kang, and Bin Wang. EvoC2F: Compiling tool orchestration for efficient and evolvable LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://openreview.net/forum?id=ZSGB91kMOG.

[32] Caleb Winston, Ron Yifeng Wang, Azalia Mirhoseini, and Christos Kozyrakis. Agent JIT compilation for latency-optimizing web agent planning and scheduling. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://arxiv.org/abs/2605.21470.

[33] Jincheng Wang, Min Zheng, and Tao Wei. COVENANT: Natural-language workflow compilation for aligned agent execution. *arXiv preprint arXiv:2607.25400*, 2026. URL https://arxiv.org/abs/2607.25400.

[34] 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*, pages 48371–48392, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

[35] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, 2023. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

[36] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

[37] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982.

[38] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

[39] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

[40] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

[41] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025. URL https://arxiv.org/abs/2504.12516.

[42] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[43] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia. Dynamo: A transparent dynamic optimization system. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 1–12, 2000. doi: 10.1145/349299.349303.

[44] Andreas Gal, Brendan Eich, Mike Shaver, David Anderson, David Mandelin, Mohammad R. Haghighat, Blake Kaplan, Graydon Hoare, Boris Zbarsky, Jason Orendorff, Jesse Ruderman, Edwin W. Smith, Rick Reitmaier, Michael Bebenita, Mason Chang, and Michael Franz. Trace-based just-in-time type specialization for dynamic languages. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 465–478, 2009. doi: 10.1145/1542476.1542528.

[45] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijałkowski, and Armin Rigo. Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems (ICOOOLPS)*, pages 18–25, 2009. doi: 10.1145/1565824.1565827.

[46] Urs Hölzle, Craig Chambers, and David Ungar. Debugging optimized code with dynamic deoptimization. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 32–43, 1992. doi: 10.1145/143095.143114.

[47] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 47–57, 1988. doi: 10.1145/73560.73564.

[48] C. K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1): 41–46, 1970. doi: 10.1109/TIT.1970.1054406.

[49] Xiaolin Zhou, Jinbo Liu, Li Li, Ryan A. Rossi, and Xiyang Hu. Counterfactual trace auditing of LLM agent skills, 2026. URL https://arxiv.org/abs/2605.11946.

[50] Headroom Labs. Headroom: Compressing tool outputs, logs, and retrieved context before they reach the LLM. Software repository, https://github.com/headroomlabs-ai/headroom, 2026. Apache-2.0; self-reported 60–95% token reduction on JSON payloads and 73% on a GitHub triage workload. Accessed 7 August 2026.

[51] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2 edition, 2016. doi: 10.1007/978-3-662-49851-4.

[52] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993. URL `https://www.itu.dk/people/sestoft/pebook/`.

What is in this appendix

The body states each result once, at the strength its evidence supports. This appendix holds the material that would otherwise interrupt that argument, in the order a reader is likely to want it. Appendix A pins the environment and gives the exact command for every study, so each number can be traced to the run that produced it. Appendix B reports test coverage and the residual engineering debt, including the compiler–runtime mismatch this review found and corrected. Appendix C is the complete claim register: every claim the work could be read as making, its direct evidence, and its verdict, including the claims the evidence does *not* support. Appendix D gives the per-metric numbers and confidence intervals behind the body's figures. Appendix E explains why each of the ten named benchmark families receives the integration depth it does, and appendix F states what the artifact's lab-only promotion did and did not demonstrate. Nothing here introduces a result that the body does not already report.

A Reproducibility Details

A.1 Environment and sealed evidence

The live run manifest records macOS 26.5.2 arm64, Python 3.14.4, OpenAI Agents SDK 0.19.2, OpenAI Python 2.52.0, `gpt-5.6-luna`, low reasoning effort, and provider default temperature because the selected model rejected an explicit temperature parameter. The result explicitly records that the OpenAI key and Hugging Face token were used and that no secret was serialized. The experimental workspace did not retain prior Git history; the public repository starts from a later initial snapshot.

The GitHub and NESTFUL revisions begin `e344be7b84d1` and `fc2c4123e735`, respectively. Their complete 40-character identifiers, exact file digests, and byte counts are in the two `source_manifest.json` files. Generated artifact digests are in the artifact manifest under `paper/results`.

A.2 Commands

The provider-free reproduction is:

```
.venv/bin/python paper/scripts/nestful_benchmark.py
R=paper/results/github_natural_replication
.venv/bin/python paper/scripts/github_live_study.py \
    --task-design natural-extractive-v2 \
    --regrade-results \
    --results-path "$R/results.json"
.venv/bin/python paper/scripts/build_artifacts.py
O=paper/results/optimizer_head_to_head
H=paper/scripts/github_optimizer_head_to_head.py
.venv/bin/python "$H" \
    --regrade-existing "$O/results.json"
M=paper/results/multidomain/preflight
V=paper/scripts/validate_multidomain.py
.venv/bin/python "$V" \
    --pool vulnerability="$M/vulnerability" \
    --pool hmda="$M/hmda" --out "$M/validation.json"
.venv/bin/coverage run -m pytest -q
.venv/bin/python scripts/verify_release.py
TECTONIC --outdir ../open_research main.tex
```

`build_artifacts.py` also writes the admission register and the cache/amortization record, so tables 1 and 9 are derived from the sealed gates and retained provider usage rather than transcribed.

The live studies are intentionally separate because they incur cost. The expanded natural-order replication, which supplies the paper's 30-pair confirmatory result, used:

```
.venv/bin/python paper/scripts/github_live_study.py \
    --task-design natural-extractive-v2 \
    --evaluation-order counterbalanced \
    --include-macro --discovery-cases 132 \
    --train-cases 16 --dev-cases 8 \
    --calibration-cases 92 --test-per-class 10 \
    --repeat-cases 10 --concurrency 8 \
    --seed 20260802
```

The prospective portfolio pilot first seals a decision from that replication and then evaluates the selected, reviewed action on previously unseen public GitHub issues:

```
P=paper/scripts/portfolio_live_study.py
.venv/bin/python "$P" \
    --preflight --cases-per-class 4
.venv/bin/python "$P" \
    --cases-per-class 4 --approve-reviewed-macro
```

The approval flag records an explicit experimental decision; it does not enable autonomous macro deployment. The script rejects overlap with every earlier study issue and writes native SDK traces plus the exact frozen decision to `results/portfolio_live`. The bounded learned/manual comparator used real public records and real provider calls:

```
H=paper/scripts/github_optimizer_head_to_head.py
.venv/bin/python "$H" \
    --train-cases 4 --val-cases 2 --test-cases 6 \
    --max-metric-calls 16 --max-proposals 3 \
    --reflection-minibatch 2
```

It executes official GEPA 0.1.4. The provider-free `--regrade-existing` command above recomputes grading, comparisons, and measurement validation without changing any retained answer, trace, token, latency, or cost observation. The earlier aggressive natural-order run used:

```
cd paper/scripts
../../.venv/bin/python \
    github_natural_workflow_study.py \
    --discovery-cases 80 \
    --train-cases 20 --dev-cases 10 \
    --calibration-cases 45 \
    --test-cases 18 --concurrency 6
```

The explicit flags below reproduce the older archived 18-pair, six-repeat prescribed-prefix design; that script's defaults select a larger run:

```
../../.venv/bin/python github_live_study.py \
    --test-per-class 6 --repeat-cases 6
```

It reads `OPENAI_API_KEY` and optionally `HF_TOKEN` from `.env`. The script never prints either value. Use `--smoke` before a paid full run.

B Implementation Audit

The package contains 19,964 measured Python statements across source, tests, demos, benchmarking, and experiment drivers; the full suite collects and passes 356 tests at 73.09% statement coverage. `pip check` and the repository release verifier pass. Coverage is concentrated where it matters: the compiler and runtime library reaches 82–91% by subpackage (entry estimation 91.2%, schemas 90.1%, execution graph 89.1%, portfolio 89.9%, registry 88.3%, capture 87.0%, runtime 84.8%, evaluation 84.9%, GRC 82.5%). The uncovered mass is study and acquisition scaffolding rather than

shipped logic: the paper's own study drivers (40.2% over 3,575 statements), the external-benchmark adapters (19.1%), the public-record pool builders (34.9%), the demonstration worlds (21.5–44.8%), and the standalone experiment runners (30.2–32.9%). Two consequences follow. Every number in this paper is produced by code in the lowest-covered tier, and the coverage total is diluted by counting that tier at all. Coverage is evidence of exercised lines, not correctness.

The review found and corrected a high-severity compiler-runtime mismatch: the miner could emit suffix-only regions while the shipped runtimes resolve at the initial boundary. `GrcConfig.prefix_only` now defaults to true; the miner records `non_prefix_runtime`; and a golden-trace regression test verifies that an otherwise eligible suffix is rejected. The archived pilot is retained as direct evidence of the impact.

Residual engineering debt includes per-fixed-candidate threshold control without candidate-family multiplicity allocation, per-artifact rather than global deployment risk control, mutable in-memory artifact objects, shared-secret rather than public-key registry signing, non-atomic directory-level registry save, no runtime concurrency scheduler, no supported multi-framework runtime adapters beyond a generic wrapper, and incomplete CI/provenance metadata in this checkout.

C Complete Claims Register

Table 13 is the register in full. Thirteen of its 24 entries — C5, C6, C7, C8, C9, C12, C13, C15, C16, C18, C22, C23 and C24 — are recorded as not supported, contradicted, not established, not implemented, or not evaluated. They are listed because a claim retired quietly tends to return; keeping them beside the claims that did survive is what stops the paper's scope from drifting after publication.

Table 13: Claim-to-evidence mapping. Unsupported claims are included to prevent later narrative drift.

ID	Claim	Direct evidence	Verdict
C1	The expected producer appears in the provenance candidate set	Pinned NESTFUL: 5,531/5,746 dependency slots	96.3% candidate recall; 80.7% unique resolution
C2	A naturally recurring prefix can remove model decisions	132 live discovery records; 30 counterbalanced real-record tests	Two-read prefix cuts requests 50%; one frozen-snapshot domain
C3	Source-grounded task quality is preserved	90/90 compiled vs. 89/90 unchanged over three families; earlier aggressive compiler: 17/18	Supported on the observed 90 pairs: no compiled-only failure bounds discordance at 3.3%; not across domains or compilation depths
C4	Configured selective admission is data hungry	NESTFUL max support 26 vs. required 92	Supported for this grid/risk/confidence setting
C5	Compaction improves text determinism	Exact-answer agreement 1/6 vs. 0/6 ($n = 6$)	Not supported
C6	The artifact is production safe	No live GitHub service, canary, or multi-domain test	Not supported
C7	The learned gate discriminates risky from safe inputs	0 positive dev examples; five of six gates admit none or all; the GCS gate admits 88/92	Not supported; the one partially selective gate is also the one whose bound exceeds the registered 5%
C8	Factual summary quality is preserved	Oracle accepts fluent fabrication	Not evaluated
C9	The learned compiler generally dominates a hand-written macro	Macro beats partial GRC on 30 pairs; GCS beats provider-visible macro on 12 later pairs	Not supported across interfaces or workflow families
C10	A separate continuation contract can recover the retained exact-source miss	Provider-free replay detects issue 6602 and checked-renders 1/18 cases	Verified counterfactual; no live latency, cost, or cross-domain claim
C11	The portfolio recommends a measured macro	30 frozen independent groups; 12 fresh paired issues	Verified on one workflow family; recommendation requires human review
C12	The portfolio synthesizes a macro or evaluates a cache action	Public portfolio API accepts externally supplied measurements only	Not implemented
C13	Portfolio selection beats an always-macro policy across workflows	One family in which the measured macro is selected	Not supported
C14	Guarded composite synthesis beats the measured provider-visible macro	12 fresh paired real-record cases; both 12/12 exact	Exploratory: requests -50%, tokens -38.9%, latency -40.0%, cost -32.3%; one workflow family
C15	GCS outperforms an equally pre-executed manual program	6 fresh paired cases; both 6/6 exact with 1 request, 1 interface, and identical input tokens	Not supported; structural parity, with no significant cost or latency difference
C16	Bounded GEPA improves this workflow	Official GEPA 0.1.4; 14 task evaluations and 3 reflection calls	Not supported; seed retained, deployment requests unchanged, optimization overhead reported separately
C17	All ten named benchmark families have an implemented disposition	10/10 source dispositions; eight reference-plan screens; five external paths executed; three use live providers	Verified, but integration depth and licensed claims differ by substrate
C18	API-Bank recurrence is sufficient for configured admission	48 candidate windows; maximum family support 8 vs. 92 required; two held-out abstentions	Contradicted; every family retires and no efficiency claim is licensed
C19	External simulated-benchmark runs are real-world demonstrations	ToolSandbox and tau evidence metadata; public simulator substrates	Not claimed; the real-record GitHub study remains the real-scenario tier
C20	A mandatory-write fulfillment workflow can be partially compacted without moving the write	Tier-3 Demo E live SDK run on fictional WMS fixtures; baseline/compacted 7.0/2.0 requests, 66.4% fewer tokens, 1.00/1.00 quality	Verified runtime boundary on the fixture; estimated cost rises 8.3%; not a production write-safety claim
C21	HMDA public records are ready for a future multidomain trace comparison	results/multidomain/preflight/validation.json	Verified provider-free preflight: 420/420 exact gold, 416/420 variable paths; zero provider calls and no optimization result
C22	Every reported artifact meets the registered 5% selective-risk target	results/admission_register.json: four artifacts at alpha=.05, three at alpha=.10	Contradicted; the GCS and earlier three-read artifacts are licensed only at 10% and would retire at 5%
C23	Token reduction is a proxy for cost reduction	results/cache_accounting.json: macro uses 30.9% fewer tokens but is 8.0% cheaper at 0.0% cache reads vs. 27.8%	Contradicted; interface fusion trades prompt length against prefix reuse and the two newer families are cache-cold throughout
C24	The reported exact gates provide compiler-wide control after candidate-family search	Per-candidate Clopper-Pearson bounds split delta over 11 thresholds; compile_grc may calibrate multiple families	Not established; freeze one candidate before calibration or adjust across candidate-threshold pairs; two candidates require 106 zero-violation groups at alpha=.05, delta=.10

Table 14: Per-metric absolute means for the prescribed-prefix ablation, the numeric backing of figs. 4 and 5. p_W is a paired two-sided Wilcoxon test over the 18 pairs; it is reported for the endpoints that vary, and the request reduction is marked deterministic because it is exactly 4 → 1 in every pair. Secondary tests are unadjusted.

Metric	Baseline	Compiled	Change	p_W
Provider requests	4.0	1.0	−75.0%	deterministic
Tool calls	3.0	3.0	−0.0%	1
Input tokens	3798.5	1294.7	−65.9%	2.0e-04
Output tokens	119.3	50.6	−57.6%	1.9e-04
Total tokens	3917.8	1345.2	−65.7%	2.0e-04
Wall latency	10.29 s	1.55 s	−85.0%	7.6e-06
Estimated cost	$0.000784	$0.000372	−52.6%	7.6e-06
Task-quality pass rate	1.000	1.000	0.0 pp	1.0

D Additional Numerical Results

The final live discovery run completes 132/132 requested episodes. Tool-contract eligibility is 132/132, task success is 129/132 (97.7%), and category accuracy is 130/132 (98.5%). Discovery consumes 517,185 input and 16,108 output tokens across 528 provider requests. All final paired-test executions complete without recorded runtime failure.

The expanded natural-order replication retains 132 discovery and 120 evaluation outputs with no infrastructure failure. Discovery produces 130/132 exact-source passes; all 132 traces use the same three-read order, while 130 enter the compiler-eligibility pool and 116 enter the 16/8/92 split. The full three-read candidate is rejected because the comment limit is ungroundable; the selected two-read program removes two provider requests. Across 30 primary pairs, its mean compiled-minus-baseline differences are: input tokens −1626.8 [−1716.2, −1541.5], output tokens −56.1 [−65.2, −47.8], total tokens −1682.9 [−1772.5, −1596.7], wall latency −3.187 s [−3.758, −2.660], and estimated cost −$0.000279 [−$0.000313, −$0.000241]. Requests change from four to two and tool calls remain three. The macro also uses two requests but one tool call, 1,780.8 tokens, 3.25 seconds, and $0.000545 per issue, versus 2,576.5 tokens, 2.97 seconds, and $0.000593 for the partial compiler. All three primary arms pass 30/30 exact-source and full task contracts. These are 10,000-sample paired bootstrap intervals; secondary tests are unadjusted.

For the 18 primary pairs, the mean compiled-minus-baseline differences and 95% paired bootstrap intervals are: input tokens −2503.8 [−2749.4, −2271.4], output tokens −68.8 [−72.1, −66.0], total tokens −2572.6 [−2816.6, −2340.2], provider latency −8.743 s [−11.741, −6.139], wall latency −8.744 s [−11.742, −6.140], and estimated cost −$0.000412 [−$0.000473, −$0.000348]. Tool-call difference is exactly zero. Table 14 gives the same run as levels rather than differences.

Two provider-free derivations accompany the primary result and are regenerated by `paper/scripts/build_artifacts.py` into `results/admission_register.json` and `results/cache_accounting.json`. The admission register reads the sealed gate of every admitted artifact. Its bounds are per fixed candidate: the compiler's search over multiple candidate families is not included in the multiplicity budget, and a simple two-candidate correction would require 106 rather than 92 zero-violation groups. Subject to that scope, the prescribed-prefix ablation and all three primary families use $\alpha = .05$ with 92 of 92 calibration groups admitted and upper bound 0.049809, while the earlier three-read natural-order artifact ($\alpha = .10$, 45/45, 0.099185) and the guarded-composite artifact ($\alpha = .10$, 88/92, 0.052013) do not meet the registered target. The cache record gives per-episode input tokens, cache-read share, and cache-write tokens for each arm: 4,062 input at 32.3% cached for the unchanged issue-type agent, 2,435 at 27.8% compiled, and 1,633 at 0.0% for the hand-written macro; both newer families record 0.0% cache reads in every arm. Paid discovery cost $0.11463, $0.09308, and $0.09882 across 132 episodes and 528 provider requests per family, giving provider-side break-even at 411, 182, and 181 future episodes for issue-type, PR-outcome, and backlog-attention routing. These exclude engineering, review, monitoring, and invalidation cost, and the issue-type figure is high because its admitted program is only two reads deep.

NESTFUL blocked-window counters are not mutually exclusive episode counts because the miner enumerates multiple candidate spans: 10,172 ambiguous-slot spans, 5,341 spans with live-ins outside the entry schema, 4,853 partial runs, and one ungrounded-slot span. The 12 synthesized families contain programs of two to five steps. The 20 failed family attempts split evenly between unsupported loop-predicate synthesis and grouped-refit grounding failure.

The portfolio selector uses the 30 independent replication issues, treating repeated conditions within an issue as one group. At 95% simultaneous confidence with separate 15% quality-risk and regret-risk limits, both the compiler and macro have zero observed violations and exact upper bounds of 0.136; their mean utilities are 0.327 and 0.489, re-

Table 15: The frozen portfolio decision and the fresh-cohort evaluation that followed it, the numeric backing of fig. 8. R_q^+/R_u^+ are the one-sided upper bounds on quality risk and regret risk under the 15% pilot limit. The lower block is a different cohort: 12 previously unseen issues, evaluated after the decision was sealed.

Calibration action	n	$\bar{U}$	R_q^+/R_u^+	Decision
Compile	30	0.327	0.136/0.136	admitted
Macro	30	0.489	0.136/0.136	selected; human review
Fresh cohort: selected macro versus unchanged				
Endpoint	n	Base	Macro	Change
Exact contracts	12	12/12	12/12	preserved
Provider requests	12	4	2	−50.0%
Tool calls	12	3	1	−66.7%
Total tokens	12	–	–	−59.2%
Estimated cost	12	–	–	−40.6%
Wall latency	12	–	–	−71.6%

Table 16: Per-condition detail for the controlled stress suite plotted in fig. 6, with the workflow shape that makes each condition hard. These are deterministic fictional fixtures executed against a live provider, not business records: they establish boundary behaviour, not field performance. Demos D, E′ and E″ are negative controls, where leaving the agent unchanged is the only correct outcome and the cost increase is the price of the guard refusing.

			Model calls		Change		Quality
Demo	Shape that makes it hard	n	base	cand.	tokens	cost	base / cand.
A	Linear read prefix	4	6.0	1.0	−79.6%	−73.6%	1.00 / 1.00
B	ACL scope + index version as guard keys	4	7.0	1.0	−76.3%	−68.7%	0.97 / 0.97
C	Coordinator with a handoff barrier	4	5.0	1.0	−73.8%	−75.5%	1.00 / 1.00
E	3 synthesized branches, pagination, mandatory write	4	7.0	2.0	−66.4%	+8.3%	1.00 / 1.00
F	Route specialization (prompt + tool surface)	4	7.0	7.0	−16.8%	+123.0%	1.00 / 1.00
D	Undeclared MCP effects (negative control)	2	3.0	3.0	+0.1%	+0.2%	1.00 / 1.00
E′	Loop-bearing artifact refused (negative control)	4	7.0	7.0	+0.2%	+55.4%	1.00 / 1.00
E″	Entry-schema drift, guard miss (negative control)	4	7.0	7.0	+0.0%	+54.5%	1.00 / 1.00

spectively. The macro is therefore recommended for human review. On 12 fresh issues, baseline and macro both pass 12/12 exact-source, factual, and tool-contract checks. The macro reduces provider requests by 50.0%, tool calls by 66.7%, total tokens by 59.2%, wall latency by 71.6%, and estimated cost by 40.6%. Paired two-sided Wilcoxon tests give $p = 0.000488$ for tokens, latency, and cost. This is a prospective within-family result, not evidence that selection outperforms an always-macro policy across heterogeneous workflows.

The bounded optimizer comparison uses four optimization-training, two validation, and six deployment issues, all disjoint from each other and every earlier cohort. Every deployment condition passes 6/6 exact factual and tool contracts. Unchanged, GEPA, GCS, GCS plus the retained GEPA seed, and manual pre-model execution use respectively 4, 4, 1, 1, and 1 mean provider requests. Mean input tokens are 3,542.0, 3,530.0, 770.8, 770.8, and 770.8. GCS and manual execution therefore tie structurally; their paired latency and estimated cost differences are not significant. GEPA retains its seed after 14 task evaluations and three reflections. Its separate optimization ledger records 59 provider requests, 63,954 tokens, and $0.01162959 estimated cost. One provider-span latency exceeds the surrounding host wall time and is retained but excluded from provider-span comparisons.

The suite in table 16 is ordered by what it tests rather than by result. The first three conditions compact cleanly. Demo E compacts a branching read prefix while leaving a mandatory write with the ordinary agent, and pays 8.3% more for it. Demo F specializes a route and costs 123.0% more, because fusing the prompt surface fragments the cache reuse that made the baseline cheap. The last three are negative controls: undeclared MCP effects, a loop-bearing candidate, and entry-schema drift each produce a refusal, and the residual cost increase is what the guard costs when it declines to act.

Table 17: Disposition of every named benchmark family. Only NESTFUL and API-Bank retain the observed intermediate values a post-trace compiler needs; the rest are interoperability evidence, and no gated path receives an imputed metric.

Benchmark	Path exercised	Scope	Observed result / boundary
NESTFUL	Compiler	1,415 traces	24 / 12 / 0 held-out; all retire
API-Bank	Compiler + API replay	212 traces	0 / 2 / 0; RETIRE; upstream 338/389 exact
BFCL v4	Official gold checker	200 tasks	200/200 valid; no model score
ToolSandbox	Live-provider subset	1 scenario	milestone 0.982; compiler not run
τ^2/τ^3	Live-provider subset	4 tasks	0/4 pass; compiler not run
ToolBench	Adapter smoke	10 fixtures	full data/backend unavailable
AgentBench	Adapter + preflight	556 tasks	services and Freebase data gated
GAIA	Access preflight	0 downloaded	authorization denied; no metric
SWE-bench Verified	Task adapter	500 issues	official run host-gated
BrowseComp	Live-web subset	3 tasks	1/3 correct; 28 searches; bypass

E Benchmark Selection Rationale

The extension now gives all ten named benchmark families an explicit path. NESTFUL and API-Bank execute the compiler because they preserve complete values. BFCL executes 200 gold plans through its official checker. ToolSandbox, the maintained τ^2/τ^3 implementation, and BrowseComp execute bounded official paths with real provider calls. ToolBench and AgentBench execute source adapters but remain full-evaluation gated by unversioned external data, live backends, and services. SWE-bench Verified's 500 real issues are normalized, while its container execution is host-gated. GAIA remains authorization gated. No missing path receives an imputed metric.

These integrations do not make the benchmarks interchangeable. A gold plan lacks observed tool results; an encrypted web question lacks a replay-safe local search trace; a stateful simulator can measure the agent but is not a real-world deployment; a real issue without an agent trajectory tests generation but not post-trace compilation. A future compiler benchmark should record entry snapshots, full argument/result payloads, effect and permission labels, independent grouping keys, commit boundaries, task outcomes, and a frozen paired protocol. Without these fields, a benchmark can measure tool use but cannot substantiate safe trace-derived replacement.

E.1 Reproduction commands for the all-source audit

Source acquisition writes into a caller-provided disposable directory and serializes only revisions, hashes, and gate reasons. After acquisition, the provider-free matrix and the API-Bank/BFCL paths are rebuilt with:

```
SCRIPTS=paper/scripts
OUT=paper/results/external_benchmarks
python "$SCRIPTS/external_benchmark_sources.py" \
  --root "$SOURCE_ROOT" \
  --output "$OUT/source_preflight.json"
python "$SCRIPTS/external_benchmark_matrix.py" \
  --source-root "$SOURCE_ROOT"
python "$SCRIPTS/api_bank_benchmark.py" \
  --source-root "$SOURCE_ROOT"
python "$SCRIPTS/bfcl_structural_benchmark.py" \
  --source-root "$SOURCE_ROOT"
```

ToolSandbox is isolated by the pinned Dockerfile. The τ and BrowseComp live runs require `OPENAI_API_KEY`; GAIA acquisition additionally checks `HF_TOKEN`. The sanitizers retain hashes, counts, scores, timing, and usage only. Raw ToolSandbox and τ trajectories remain in disposable upstream run directories, and BrowseComp questions and answers are decrypted only in process memory.

The repository now contains the provider-free portion of such a prospective extension. It checksum-binds 420 real vulnerability groups and 420 privacy-modified public HMDA groups, independently reconstructs exact gold, freezes group/lineage-aware study roles, and supplies hash-chained execution and sealed-analysis controls. SEC acquisition remains gated on a compliant source contact, so the three-domain protocol cannot yet freeze and no provider, quality,

latency, cost, or determinism result exists. The retained preflight is feasibility evidence, not a third empirical tier in this manuscript.

F Artifact Lifecycle Boundary

The study artifact passed replay and calibration, then was activated only under `paper-protocol-lab-only` to measure its runtime path. That flag is what makes the runtime numbers in section 5.3 observations of a real dispatch rather than a simulation, and it is also the limit of what they cover.

Nothing in this study bears on approval, shadow or canary health, rollback, or monitoring: no such condition was run, so the paper reports no evidence for or against the artifact's behaviour under any of them. The same holds for privacy, tenant isolation, incident response, and drift. Readers should treat every result here as measured under a single lab-only activation of one artifact, and treat the operational questions as untested rather than as satisfied.