CODE-TO-HARNESS: LEARNING LANGUAGE POLICIES THROUGH EXECUTABLE PRACTICE

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning reusable policies for frozen large language models (LLMs) raises a representation question: must a policy be discovered in the same form in which it is deployed? We introduce Code-to-Harness, which uses executable programs as the discovery representation and compact instructions as the deployment representation. A practice agent proposes and revises optimizer programs from executable feedback; a development competence screen determines which completed runs are eligible for promotion; and a one-shot in-context synthesis compresses the accumulated executable practice into an approximately 200-word frozen policy. Only policies from promoted runs enter the primary transfer evaluation, and no source program is executed at deployment. Across ten completed discovery runs, four satisfy the recorded promotion criterion. On 30 genuinely fresh objective instances per task, promoted policies reduce mean regret relative to the unaided executor by 44.4–60.7% on quadratics and 32.5–50.0% on Rastrigin. One fixed policy attains lower mean regret than all five numerical baselines on eight of ten BBOB families, and all four promoted texts improve a second vendor's executor on both core tasks. On three LCBench/YAHPO-Gym HPO tasks, frozen policies attain the lowest reported mean on two datasets. A five-batch audit further shows that the recorded threshold behaves as a conservative competence screen rather than a calibrated predictor of downstream text quality. Together, the results show that useful search-strategy information can cross an executable-to-language representation boundary without requiring the discovery program at deployment.[1]

1 INTRODUCTION

A frozen large language model (LLM) can act as a black-box optimizer by observing evaluated points, proposing the next query, and adapting to feedback. Its behavior is strongly shaped by the strategy text it receives, which raises a basic policy-learning question: *must a reusable policy be discovered in the same representation in which it is ultimately deployed?*

Most prompt-optimization and memory methods use language for both stages (Pryzant et al., 2023; Agrawal et al., 2026; Zhang et al., 2026a; Suzgun et al., 2025; Zhang et al., 2026b). Language is a natural interface for a frozen LLM, but it is a comparatively loose discovery object: textual advice need not commit to concrete actions until an executor interprets it, and evaluating each candidate requires repeated LLM execution. An optimizer program makes the opposite tradeoff. It commits to explicit query decisions, can be executed and scored directly, and can be replayed over development objectives without invoking an LLM at every optimization step.

This motivates CODE-TO-HARNESS: *discover in code, deploy through language*. During development, a practice agent proposes optimizer programs and revises them from executable outcomes. A development competence screen determines which completed runs have sufficient source-side evidence for promotion. The practice agent then compresses its accumulated executable-practice experience into a compact textual policy. At evaluation time, only texts from promoted runs are used: the policy and executor configuration are frozen, the observation history evolves, and the source program is no longer invoked. The central contribution is therefore the *discovery–deployment representation split*: executable behavior supplies a precise learning target, while text supplies the interface consumed by the frozen LLM.

[1] Code: https://anonymous.4open.science/r/code-to-harness-3608/README.md.

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

Shifted quadratics serve as a controlled, replayable practice environment rather than as a claim that LLMs should replace classical optimizers on low-dimensional numerical problems. Generalization is tested on genuinely fresh objective instances, Rastrigin, ten BBOB families, larger budgets, $d = 20$, additional LLM executors, and three public HPO tasks served by YAHPO-Gym (Zimmer et al., 2021; Pfisterer et al., 2022). Across ten completed discovery runs, the recorded promotion decisions identify A, B, v11, and v14 as passes and the other six as failures; all four promoted texts are then carried unchanged into transfer evaluation. Across the four texts, recurring motifs include information-efficient probing, incumbent-centered adaptation, directional exploitation, and budget-aware refinement, while implementations and transfer profiles remain distinct. On fresh core-task instances the four texts retain their gains; one fixed text is lower than all five numerical baselines on eight of ten BBOB families; all four improve a second vendor's executor on both core tasks; and frozen policies attain the lowest reported mean on two of three LCBench datasets. Failures at higher dimension and on Flash-Lite provide corresponding boundary cases.

The experiments therefore separate three questions: whether executable practice can produce a useful frozen text policy, whether that policy transfers, and whether the promotion screen behaves consistently as a development evidence filter. Numerical optimizers provide competitiveness context, while language-based controls probe alternative discovery or deployment routes under the implemented settings. The release contains the exact frozen A/B/v11/v14 texts and hashes; archived champion programs and practice logs further link B, v11, and v14 to their source-side runs. Overall, the evidence supports a focused claim: useful strategy information need not be learned in its deployment representation.

2 Related work

Prompt optimization and adaptive context. ProTeGi, GEPA, EvoPrompt, and SPRIG revise prompts or system instructions from execution feedback (Pryzant et al., 2023; Agrawal et al., 2026; Guo et al., 2024; Zhang et al., 2026a). Reflexion, Dynamic Cheatsheet, and Agentic Context Engineering instead retain and curate linguistic experience (Shinn et al., 2023; Suzgun et al., 2025; Zhang et al., 2026b). These approaches keep discovery largely in language. Code-to-Harness asks a different representation question: can strategy information first be discovered in executable form and only then be transferred into text for frozen deployment? Our adapted text-space controls probe this question under the implemented settings rather than ranking entire method families.

Executable discovery and agent search. FunSearch evaluates generated programs through execution (Romera-Paredes et al., 2024); CALM couples algorithm generation with reinforcement learning (Huang et al., 2026); AFlow searches code-represented workflows (Zhang et al., 2025b); and the Darwin Gödel Machine maintains an archive of self-modifying agents (Zhang et al., 2025a). Code-to-Harness shares executable feedback but changes the deployment artifact: executable practice is summarized into instructions, and only texts from screened, promoted runs are used at test time; the source program itself is not executed. The novelty claim is therefore the discovery–deployment representation boundary, not executable program search in isolation.

LLMs for black-box optimization. OPRO generates candidates from scored histories (Yang et al., 2024); LLAMBO uses LLMs for warm-starting, surrogate modeling, and candidate sampling (Liu et al., 2024); LMABO selects acquisition functions online (Ngo et al., 2026); and Bayesian-OPRO uses Bayesian optimization to select conditioning histories (Agarwal et al., 2025). These methods couple LLM inference to optimization episodes. Our learned component is a fixed text acquired beforehand, so numerical and LLM-mediated baselines serve as context for deployment-time competitiveness rather than defining the contribution.

3 Learning language policies through executable practice

Code-to-Harness separates the representation used for policy discovery from the representation used for deployment. Programs are used during discovery because their query decisions are executable, replayable, and directly scoreable. Text is used at deployment because the target is a frozen LLM conditioned through instructions and observation history. During development, an

agent proposes optimizer programs, revises them from executable feedback, and records the resulting trajectories. A development competence screen determines which completed runs are eligible for promotion, while a one-shot synthesis compresses the accumulated executable-practice context into a compact language policy. The source program and the text-induced policy are evaluated separately; distillation transfers strategy information rather than compiling the program into an exact numerical replica.

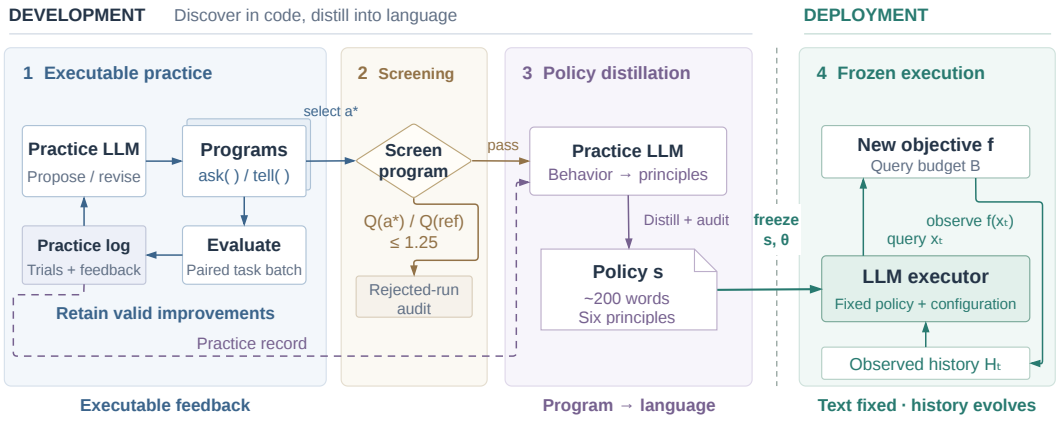

Figure 1: **Discover in code, deploy through language.** Executable practice produces candidate optimizer programs and an accumulated practice context; a development screen determines promotion eligibility, and one-shot synthesis compresses the practice experience into a frozen language policy. Only promoted texts enter the primary transfer evaluation. At deployment, the text and executor configuration are fixed and only the observation history evolves.

3.1 Problem formulation: learning a reusable search policy

Consider minimizing an unknown objective $f : \mathcal{X} \to \mathbb{R}$ on a bounded domain $\mathcal{X} \subset \mathbb{R}^d$. After t evaluations, the optimizer observes $H_t = ((x_i, y_i))_{i=1}^t$ with $y_i = f(x_i)$ and proposes the next point. With task budget B, performance is final simple regret,

$$r_B(\pi; f) = \min_{1 \le t \le B} f(x_t) - f^\star, \qquad f^\star = \min_{x \in \mathcal{X}} f(x). \tag{1}$$

Only evaluated points contribute to the metric, and the optimizer is not given $f^\star$ or the instance parameters.

A strategy text s and fixed executor configuration θ induce a proposal policy $\pi_{s,\theta}$. Here θ includes the model, history formatter, output parser, and inference settings. Our target is a reusable s with low regret on new objectives, with θ fixed within each comparison. Code-to-Harness searches over executable policies on a development distribution and then transfers the selected strategy into text. Neither model weights nor s are updated during a test episode.

3.2 Executable practice for policy discovery

A candidate program implements a sequential proposal–feedback interface. The practice agent maintains an incumbent a_{r-1} and record $\mathcal{L}_{r-1}$; at round r it proposes $\widetilde{a}_r = \mathrm{Propose}(a_{r-1}, \mathcal{L}_{r-1})$. Candidates must make concrete query decisions—for example directions, step sizes, and transitions between search phases—so those decisions can be executed and compared directly.

On a development batch $\mathcal{P}_r$ of m instances, candidate and incumbent use the same task budget B_{dev}:

$$Q_r(a) = \frac{1}{m} \sum_{f \in \mathcal{P}_r} r_{B_{\mathrm{dev}}}(a; f), \qquad a_r = \begin{cases} \widetilde{a}_r, & \text{if valid and } Q_r(\widetilde{a}_r) < Q_r(a_{r-1}), \\ a_{r-1}, & \text{otherwise.} \end{cases} \tag{2}$$

The record stores source, outcomes, trajectories, retention decisions, and diagnoses for later proposals. The reported protocol uses $R = 10$ rounds, $m = 16$, $d = 8$, and $B_{\mathrm{dev}} = 20$ on shifted

Algorithm 1 CODE-TO-HARNESS: executable discovery and language-policy deployment

Require: Development batches $\{\mathcal{P}_r\}_{r=1}^{R}$; common selection batch $\mathcal{V}_s$; screen data $\mathcal{V}_g$; reference a_{ref}; template τ; executor configuration θ

Initialize incumbent a_0 (or $\varnothing$), record $\mathcal{L}_0$, and valid-program archive $\mathcal{C} \leftarrow \varnothing$
for $r = 1, \ldots, R$ **do**
  $\widetilde{a}_r \leftarrow \mathrm{Propose}(a_{r-1}, \mathcal{L}_{r-1})$
  Check validity; evaluate valid candidate and any incumbent on $\mathcal{P}_r$
  Set a_r by Equation 2; initialize if no incumbent exists
  Append the attempt and feedback to $\mathcal{L}_r$; archive valid programs
end for
$a_\star \leftarrow \arg\min_{a \in \mathcal{C}} Q_{\mathcal{V}_s}(a)$
$g \leftarrow \mathbb{I}[Q_{\mathcal{V}_g}(a_\star)/Q_{\mathcal{V}_g}(a_{\mathrm{ref}}) \le 1.25]$
$s \leftarrow \mathrm{Distill}_\tau(\mathcal{L}_R)$ ▷ one in-context synthesis; six-principle text
Apply the documented text audit to any artifact considered for promotion
if $g = 0$ or the text audit fails **then**
  return rejected-run record; retain s only as a diagnostic artifact
end if
$A_F \leftarrow \mathrm{Seal}(s, \theta)$
return A_F ▷ freeze before evaluation on new objectives

quadratics, while restricting programs to elementary operations on the observation history rather than fitting, kernel, or optimizer libraries.

After practice, the released protocol rescored all valid round programs on a common 40-instance development validation set and selected the lowest-mean program as the champion $a_\star$. This is distinct from the within-round incumbent update in Equation 2. For the nine runs with recovered round logs (B, C, and v8–v14), the archived champions are consistent with this rule; the remaining provenance boundary is documented in Appendix A.3.

3.3 DISTILLING EXECUTABLE STRATEGIES INTO LANGUAGE POLICIES

After practice, a fixed instruction τ asks the same practice agent, within its accumulated conversation, to summarize the executable experience as six operational principles,

$$s = \mathrm{Distill}_\tau(\mathcal{L}_R). \tag{3}$$

Here $\mathcal{L}_R$ denotes the accumulated practice context: proposed programs, execution outcomes, trajectories, retention decisions, and diagnoses. The champion is selected and screened independently; it is not re-injected as a new message before the historical in-context synthesis. Appendix B.3 therefore provides a *descriptive* source-to-text alignment, while Appendix B.2 gives a standalone fresh-conversation reconstruction for explicit code/record distillation and the controlled ablations.

The reported promoted texts satisfy the documented 150–250-word, six-principle format and content screen. The primary evaluation does not search over alternative syntheses using test outcomes. Distillation here is prompt-based synthesis, not parameter training or guaranteed numerical emulation. Controlled ablations test how program structure, program explanation, and the practice record can carry information into this stage (Appendix D).

3.4 PROMOTION SCREEN AND FROZEN DEPLOYMENT

For promotion, the champion is compared with a fixed reference coordinate-search program a_{ref} (denoted *evo*; implementation identifier in the appendix) on a development screen batch $\mathcal{V}_g$:

$$G(a_\star; \mathcal{V}_g) = \mathbb{I}\left[\frac{Q_{\mathcal{V}_g}(a_\star)}{Q_{\mathcal{V}_g}(a_{\mathrm{ref}})} \le 1.25\right]. \tag{4}$$

The historical 1.25 threshold defines a minimum executable-competence boundary for the source program; it is not a ranker or predictor of downstream text quality. Later clean-seed and multi-batch analyses characterize decision stability without changing historical promotion decisions (Appendix H.1). The released runner can still emit a one-shot text for a rejected completed run; such texts are retained only for diagnostics and never enter the primary transfer evaluation.

After text audit, we freeze $A_F = \mathrm{Seal}(s, \theta)$. A formatter ϕ_θ supplies the task interface and observation history to the LLM, and a parser ψ_θ converts the response into a bounded query:

$$z_{t+1} \sim M_\theta(\cdot \mid s, \phi_\theta(H_t, \mathcal{X})), \qquad x_{t+1} = \psi_\theta(z_{t+1}) \in \mathcal{X}. \tag{5}$$

Within a Base–Harness comparison, the executor configuration is shared and the frozen strategy text is the intervention. Cross-executor tests keep the text fixed and change the executor configuration. Across the recorded linear study, four of ten completed runs are promoted; A, B, v11, and v14 are accepted realizations of the same pipeline, not four hand-designed algorithms, and no task-specific post-hoc artifact selection is used.

4 Experiments

We evaluate three questions: whether executable practice yields a useful frozen text policy, whether that policy transfers beyond the practice setting, and how the promotion screen behaves as a development evidence filter. The recorded campaign contains ten completed linear discovery runs. Recorded promotion decisions identify A, B, v11, and v14 as passes and C, v8–v10, and v12–v13 as failures; the released source-side artifacts reproduce those decisions for the nine runs with archived champions. All four promoted texts are reported rather than selected by held-out transfer performance. Transfer results therefore characterize quality conditional on promotion, not the expected outcome of an arbitrary discovery attempt.

4.1 Evaluation protocol

Core tasks are shifted quadratics and Rastrigin at $d = 8$, $B = 20$, with $N = 30$ instances per setting. The quadratic family is the controlled practice environment; transfer adds ten BBOB families (Finck et al., 2009), budgets 50/100, $d = 20$, additional executors, and three LCBench HPO tasks through YAHPO-Gym (Zimmer et al., 2021; Pfisterer et al., 2022). Methods, texts, and thresholds are frozen before a 30-instance confirmation on genuinely distinct objective instances; LCBench is surrogate-served from real training runs.

Base is the unaided executor with the same interface, so Base versus Harness isolates the frozen strategy text. GP-BO, CMA-ES, HEBO, TPE, SMAC3, and random search provide numerical context where available (Snoek et al., 2012; Hansen, 2016; Cowen-Rivers et al., 2022; Bergstra et al., 2011; Lindauer et al., 2022). OPRO-style and LLM+BO controls probe alternative LLM-mediated routes under task-adapted implementations. We denote the fixed screening reference by *evo*. Exact frozen texts and SHA-256 identities are released for all four promoted policies; champion programs and full practice logs are additionally available for B, v11, and v14. The historical validation and G2 ranges overlap one practice round by 16 seed labels each, so both are treated strictly as development data; an untouched-seed gate re-audit is reported separately. Configuration, provenance, test coverage, and multiplicity details are in Appendix E.

4.2 Policy gains persist on genuinely fresh objective instances

On fresh instances, all four promoted policies beat Base on both tasks (Table 1), reducing mean regret by 44.4–60.7% on quadratics and 32.5–50.0% on Rastrigin, with paired $p \leq .001$ in all eight comparisons. B is the lowest text on quadratics (21.9) and A on Rastrigin (43.1). The reference remains lowest on quadratics (16.9), while every frozen text is lower on Rastrigin (43.1–58.1 versus 74.7), preserving the off-family reversal on genuinely new objectives.

The multi-batch audit also separates promotion evidence from downstream text quality: v11/v14 pass all five fresh screen batches, whereas B passes 2/5 despite producing the strongest quadratic text. Applied post hoc to five-batch mean ratios on ten text-bearing runs, the historical $\tau = 1.25$ threshold has sensitivity 0.38 and specificity 1.0. We therefore interpret it as a conservative competence screen, not a calibrated text quality predictor (Appendix H.2).

Comparator context. On the original core instances, all four texts are below HEBO, Optuna-TPE, and SMAC3; the reference remains lowest on quadratics and v11 on Rastrigin (Table 11). The original OPRO-style control is worse than Base, whereas strengthened OPRO+ reaches 63.0/89.4; no statistically significant difference from Base is detected in the reported paired tests ($p = .556/.237$),

Method	Quadratic			Rastrigin		
	Original	Fresh	p vs Base	Original	Fresh	p vs Base
Base	69.5	55.8	–	77.4	86.2	–
Harness A	36.0	28.0	1×10^{-3}	65.3	**43.1**	2×10^{-5}
Harness B	21.4	21.9	6×10^{-7}	57.0	55.3	2×10^{-7}
Harness v11	30.5	28.7	6×10^{-4}	**49.2**	46.4	7×10^{-7}
Harness v14	24.2	31.1	7×10^{-4}	59.9	58.1	3×10^{-5}
Reference program (evo)	**18.5**	**16.9**	5×10^{-8}	83.8	74.7	.109
GP-BO	30.4	37.0	.067	83.9	89.7	.792

Table 1: **Confirmation on 30 genuinely fresh objective instances per task.** Mean final regret; fresh-set p-values are paired Wilcoxon tests vs. Base; bold marks minima. Original and fresh objective sets are distinct.



Figure 2: **Each frozen text transfers differently.** Mean regret relative to the matching Base on ten BBOB families ($d = 8$, $B = 20$, $N = 30$); values below one favor the frozen policy. No family-specific policy selection is used.

while every promoted policy remains lower in mean. LLAMBO-style reaches 110.5/129.2 on the core tasks, and LMABO-style is evaluated on LCBench. These task-adapted controls provide context rather than method-family rankings (Appendix F.2–F.3).

4.3 Transfer across objectives, budgets, and executors

Objective families. A, B, v11, and v14 improve over Base in 9, 8, 7, and 9 of the ten BBOB families and are lower than all five numerical baselines in 5, 8, 5, and 7 families, respectively. For B, four of the ten comparisons against the lowest-mean numerical baseline remain below .05 after Bonferroni correction. The transfer is not uniform: all four texts are worse than Base on Katsuura, and GP-BO is lower than every text on Attractive Sector and Schaffers F7 (Appendix L).

Budget and dimension. At $B = 100$ on Rastrigin, v11 reaches 6.8 versus 53.2–55.2 for evo, HEBO, and GP-BO, with reported paired $p < 10^{-8}$ against each. On quadratics, v14/v11 reach 3.1/3.6, close to evo's 3.8, while HEBO is lower at 0.79. At $d = 20$, v11 and v14 remain below Base on both tasks, whereas B is worse than Base by 35.5% on quadratics and 13.6% on Rastrigin. These separately configured endpoints show both longer-budget gains and a clear higher-dimensional failure mode (Appendix M).

Cross-executor transfer. All four texts improve Sonnet on both tasks (Table 2), reducing quadratic regret by 50.6–78.3% and Rastrigin regret by 24.0–50.5%; all eight reported comparisons remain below .05 under an eight-test Bonferroni correction. B reaches 24.3/54.4 on Sonnet versus 21.4/57.0 on Flash, despite Sonnet's much weaker unaided means. Flash-Lite shows smaller Rastrigin gains (4.3–11.3%) and only partial test coverage, indicating that instruction-level transfer depends on the executor (Appendix K).

Public HPO transfer. On three LCBench/YAHPO-Gym tasks ($B = 20$, $N = 30$), HEBO has the lowest reported mean on OpenML 3945, while v14 and A have the lowest reported means on

Executor	Task	Base	A	B	v11	v14
Sonnet	Quadratic	112.2	46.0	24.3	55.4	55.0
Sonnet	Rastrigin	109.8	83.5	54.4	71.1	79.0
Flash-Lite	Quadratic	112.9	51.8	79.8	99.2	105.8
Flash-Lite	Rastrigin	115.1	108.3	110.1	107.4	102.1

Table 2: **Frozen texts on additional executors.** Mean final regret on 30 paired instances. All eight Sonnet comparisons have reported $p \leq 10^{-4}$ versus the matching Base; Flash-Lite tests are partially reported.

Task	Full	Code-only	Degradation	Paired p
Quadratic (practice family)	24.4	33.4	+37%	.017
Rastrigin (off-family)	30.6	47.5	+55%	.026

Table 3: **Harness A mechanism study.** Full supplies the practice record in addition to the bare program; Code-only omits it. Two generations per condition, Pro executor, $N = 30$ paired instances.

126026 and 167190. Reported paired comparisons show that A improves its unaided executor on all three; B and v11 are negative on 167190. LCBench is more application-proximal than the synthetic core tasks but remains surrogate-served (Appendix G).

4.4 Discovery routes and distillation carriers

Alternative discovery routes. No-code notebook practice produces four frozen variants from two chains; seven of eight task means are worse than Base, and every variant is worse than every promoted Code-to-Harness text on both core tasks. Four ProTeGi-style chains accept no update under the conservative rule; under a retrospective relaxed criterion, 11/40 rewrites would be accepted, but none enters the promoted-policy range on held-out quadratics. In the single GEPA-style run, all ten mutations are worse on development data than its informed Harness-A-derived initialization. These controls are implementation-specific probes of text-space discovery, not general rankings of the corresponding method families (Appendices I.1–I.3).

An archive-based executable-search variant provides a complementary check: one of two runs originally passes the historical screen and yields 26.0/57.3 on quadratic/Rastrigin; both programs pass the later clean-seed re-audit. Two runs are too few to establish a reliability advantage, but the result shows that the code-to-language interface is compatible with a different executable search procedure (Appendix J). Code practice uses 11–15 LLM calls per completed run versus 3,210–6,420 for the tested language controls because candidate evaluation runs code; this is a model-call accounting result, not a claim about total compute, tokens, fees, or end-to-end cost (Appendix E.5).

4.5 Ablating distillation inputs

Harness A provides the cleanest record-channel test because its champion is bare code without explanatory comments or docstrings. Both conditions therefore inherit the same practiced program and differ only in whether the separate practice record is supplied to the distiller.

Removing A's record raises regret by 37% on quadratics and 55% on Rastrigin in the two-generation study (nominal paired $p = .017/.026$), but these become $.102/.130$ under the exploratory eight-comparison Holm adjustment. We therefore treat A as a mechanism exhibit rather than population-level proof. Complementary champion ablations show heterogeneous carriers: B remains near Full when the separate record is removed but degrades under Record-only, while v11 degrades when both record and explanatory comments are stripped. The combined evidence supports a multiple-carrier view in which practice-derived information may enter distillation through program structure, explanatory text, or the separate practice record (Appendix N).

5 Discussion and limitations

What crosses the representation boundary. The central result is functional representation transfer, not behavioral equivalence. The executable reference leads on practiced quadratics, whereas every promoted text is lower on Rastrigin and BBOB yields distinct family profiles. Across A/B/v11/v14, recurring motifs include information-efficient probing, incumbent-centered adaptation, directional exploitation, and budget-aware refinement. The overlap is in reusable search principles rather than identical trajectories; Code-to-Harness is therefore an abstraction pipeline, not a code-to-text compiler.

Why separate discovery from deployment. Programs commit to concrete actions that can be replayed and scored without invoking an LLM at every development evaluation; text is used at deployment because the target interface is a frozen instruction-following LLM. The value of the split is interface reuse, not a claim that language execution is uniformly more accurate, cheaper, or safer than running a program directly. Sonnet transfer shows reuse across executors, whereas Flash-Lite and $d = 20$ failures show that the effect is executor- and regime-dependent.

Promotion, selection, and evidence level. Four of ten completed runs are promoted, so transfer tables measure quality *conditional on promotion*, not the outcome of an arbitrary discovery attempt. The same development rule is applied before BBOB, executor, or HPO evaluation; no family-specific text is selected afterward. The 1.25 threshold is therefore a competence screen, not a ranker of frozen texts. Its five-batch audit shows high observed specificity but low sensitivity on a small labeled sample, supporting a conservative screening interpretation while leaving threshold calibration unresolved. Numerical and LLM-mediated comparators provide context rather than method-family rankings.

Where practice information is carried. The ablations argue against a single mandatory carrier. A degrades when its separate practice record is removed; B remains near Full without that record; and v11 worsens when both comments and the record are stripped. Practice-derived information can thus reach distillation through program structure, explanatory text, or the explicit record. Likewise, the 11–15 versus 3,210–6,420 LLM-call comparison isolates one computational advantage of executable development evaluation, but does not establish lower total compute, tokens, fees, or end-to-end cost.

How the evaluation blocks fit together. The experiments form an evidence hierarchy rather than a single leaderboard: fresh objective instances test persistence on the core interface; BBOB, budget, and dimension probe objective and regime shift; cross-executor and HPO blocks probe transfer; and discovery-route controls probe alternative search representations. The contribution is the repeated usefulness of fixed promoted artifacts across these changes, together with explicit counterexamples where transfer weakens or fails.

Limitations and next steps. Exact texts and hashes are available for A, B, v11, and v14; B/v11/v14 also link to champion programs and full practice logs. Remaining provenance gaps are A's source-side run trace, byte-exact synthesis API metadata, and some endpoint/decoding manifests. With only two distillation generations per condition, A's nominal effects do not survive the exploratory Holm adjustment and remain mechanism evidence rather than population-level proof. LCBench is surrogate-served, while Appendix O is external-validity context rather than core frozen-transfer evidence. Future work should increase run- and distillation-level replication, test higher-dimensional and language-rich tasks, and report full token, latency, and monetary cost.

6 Conclusion

Code-to-Harness separates how a policy is discovered from how it is deployed. Programs make candidate search behavior executable and scoreable; language carries selected strategy information into a frozen LLM interface. Across the reported campaign, this representation transfer remains useful across new objective instances, objective families, executors, and HPO tasks, while exposing clear failure modes and provenance limits. Reusable language policies therefore need not be discovered in language.

REPRODUCIBILITY STATEMENT

Algorithm 1 specifies the intended discovery and deployment pipeline, and the appendices separate artifact-level reproducibility from byte-exact historical replay. The released package contains all four promoted frozen texts (A, B, v11, v14) with SHA-256 identities, the executable practice/evaluation runners, study manifests, and archived source-side artifacts; B, v11, and v14 are linked to champion programs and full practice logs. The source release does not bundle provider-backed per-trial evaluation caches. Instead, the manifests and runners specify how to regenerate study outputs in a configured provider environment, while the manuscript preserves the reported aggregate tables.

Exact historical replay of every LLM-mediated call is narrower. Appendix E.4 lists the remaining gaps, principally the original source-side trace for A, byte-exact historical synthesis API requests and response metadata, and some run-specific endpoint/decoding manifests. The standalone synthesis request in Appendix B.2 is a reconstruction for replication rather than the archived API envelope of every reported run. We therefore distinguish released artifact identity from unavailable call-level provenance. An anonymized repository link will be supplied with the submission materials. The production case in Appendix O is aggregate-only and is outside the exact-replay claim.

ETHICS STATEMENT

The main benchmark study evaluates synthetic numerical objectives and surrogate-served HPO benchmarks derived from real neural-network training runs. Appendix O additionally reports aggregate outcomes from a production traffic tuning experiment on an anonymized commercial recommendation platform. The manuscript reports no individual-level user data or demographic attributes and releases no production traffic logs. Its principal concerns are reliable execution of generated programs, faithful reporting of aggregate evidence, and avoiding over-interpretation of the historical production comparison. Program validity checks do not establish security for running generated code in unrestricted environments, and the reported benchmarks or production case should not be treated as validation for safety-critical deployment.

AI USE STATEMENT

LLMs are components of the proposed method and the language-based controls: they propose optimizer programs, synthesize or revise textual strategies, and execute optimization queries as specified in the method and experimental protocols. Generative AI assistance was also used to develop the manuscript's structure and wording, interpret reported results, identify literature, and create or edit LaTeX, data-processing scripts, and scientific figures. It also provided feedback on research methodology and experimental protocol design and assisted with code and implementation review. Empirical values were checked against the experimental summaries; AI-assisted manuscript preparation did not generate or alter the empirical observations reported in this paper. All AI-assisted suggestions, prose, code, figures, and analyses were independently reviewed by the authors. Responsibility for the final submission, including its claims and artifacts, rests with the authors.

REFERENCES

Dhruv Agarwal, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. Searching for optimal solutions with LLMs via bayesian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a79237d6c47feef24c1df723cb41f674-Abstract-Conference.html.

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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011. URL https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.

Alexander I. Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Jianye Hao, Jun Wang, Jan Peters, and Haitham Bou-Ammar. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022. doi: 10.1613/jair.1.13643. URL https://jair.org/index.php/jair/article/view/13643.

Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009. URL https://coco-platform.org/testsuites/bbob/overview.html. Updated version as of February 2019.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/9156b0f6dfa9bbd18c79cc459ef5d61c-Abstract-Conference.html.

Nikolaus Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016. URL https://arxiv.org/abs/1604.00772.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. URL https://www.jstor.org/stable/4615733.

Ziyao Huang, Weiwei Wu, Kui Wu, Wei-Bin Lee, and Jianping Wang. CALM: Co-evolution of algorithms and language model for automatic heuristic design. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/75ed738c083f8941cfc990245e4f6265-Abstract-Conference.html.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL https://www.jmlr.org/papers/v23/21-0888.html.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/84b8d9fcb4e262fcd429544697e1e720-Abstract-Conference.html.

Giang Ngo, Dat Phan Trong, Dang Nguyen, Sunil Gupta, and Svetha Venkatesh. Adaptive acquisition selection for bayesian optimization with large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f1b7a29fe9759d4c9d23f78738e40b42-Abstract-Conference.html.

Florian Pfisterer, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. YAHPO Gym – an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pp. 3/1–39, 2022. URL https://proceedings.mlr.press/v188/pfisterer22a.html.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://www.nature.com/articles/s41586-023-06924-6.

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

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://arxiv.org/abs/1206.2944.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025. URL https://arxiv.org/abs/2504.07952.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.03409.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a. URL https://arxiv.org/abs/2505.22954.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Lechen Zhang, Tolga Ergen, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. SPRIG: Improving large language model performance by system prompt optimization. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/2c9d78ed62ff5bf2377c3840188114c0-Abstract-Conference.html.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://arxiv.org/abs/2510.04618.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43 (9):3079–3090, 2021. doi: 10.1109/TPAMI.2021.3067763.

A ALGORITHM DETAILS AND PROTOCOL SCOPE

This appendix describes the practice, distillation, and execution stages of CODE-TO-HARNESS. Algorithm 1 specifies one discovery run; repeated runs evaluate discovery variability without selecting policies on test outcomes. The implementation details and verbatim policy texts below are recovered from the replication package. Appendix B.2 separately supplies a complete reconstructed synthesis request; it is not an archived request from the reported runs. Their scope, and the additional records needed to reproduce every evaluation, are specified in Appendix E.4.

A.1 OBJECTS AND BUDGETS

Symbol	Meaning
$f, \mathcal{X}, d$	Objective, bounded search domain, and dimension.
B	Number of objective evaluations available in one test episode.
R, m, B_{dev}	Practice rounds, paired instances per round, and queries per practice episode.
$a_r, \widetilde{a}_r$	Incumbent after round r, and the candidate proposed in that round.
$\mathcal{L}_r, \mathcal{C}$	Practice record and archive of valid candidate programs.
$\mathcal{V}_s, a_\star$	Common development selection set and its selected champion program.
$\mathcal{V}_g, a_{\mathrm{ref}}$	Gate instances and fixed reference optimizer.
T, s, τ	Descriptive program-to-principle mapping, deployment text, and distillation specification.
θ, A_F	Executor configuration and frozen text–configuration pair.

Table 4: Notation distinguishes the optimizer program from the language policy and distinguishes development resources from the budget of a single task.

For a complete valid round with an incumbent, paired screening uses at most $2mB_{\mathrm{dev}}$ objective evaluations, excluding validity checks or separate champion-selection evaluations. The corresponding upper bound for R such rounds is $2RmB_{\mathrm{dev}}$, or 6,400 with $R = 10$, $m = 16$, and $B_{\mathrm{dev}} = 20$. This is an accounting bound, not a measured total cost: initialization, invalid attempts, validation, and reuse of computed evaluations affect the total. Evaluating both the champion and reference on n_g gate instances requires an additional $2n_gB_{\mathrm{dev}}$ calls when neither arm is cached. Code execution still has computational cost even when it requires no model calls.

Policy length varies across artifacts: Harness A contains 197 whitespace-separated words and Harness B contains 205. The text audit permits 150–250 words. Appendix B.4 includes all four promoted texts.

A.2 PRACTICE OBJECTIVES AND PROGRAM INTERFACE

The quadratic generator in the replication package draws $c \sim \mathrm{Uniform}([-5, 5]^d)$ and a matrix $W \in \mathbb{R}^{d\times d}$ with independent entries in $[-1, 1]$, then defines

$$f(x) = (x - c)^\top W^\top W(x - c), \qquad \mathcal{X} = [-5, 5]^d. \tag{6}$$

Its optimum value is zero. Instance parameters are held by the evaluator; optimizer decisions use observed points and objective values.

Practice programs implement `Opt(dim, budget, rng)`, with `ask()` returning a point and `tell(x, y)` ingesting its evaluated value. The evaluation package exposes the analogous `next_batch()` / `update(points, values)` interface. The principal configuration is sequential, with batch size one. The program validity specification requires a complete 20-evaluation episode without an exception, bounded proposals, and deterministic behavior conditional on its random-number generator. The validity protocol allows up to three consecutive failed attempts before a practice round is forfeited.

The practice specification requires elementary arithmetic on the observation history, without calls to fitting, kernel, or optimizer libraries. This permits operations such as finite differences and explicitly implemented one-dimensional interpolation; it does not prohibit numerical arrays. The plain-language explainability instruction is a design constraint on generated programs, not a verified semantic property of every execution.

A.3 Within-round retention and champion selection

Each recovered linear-run practice log contains ten rounds, with each challenger evaluated on sixteen paired quadratic instances. Retention follows the lower-mean comparison in Equation 2; because successive rounds use different batches, the final incumbent is not automatically the run champion.

The released runner therefore applies a common development-side selection step after practice: every valid round program is rescored on the 40-instance validation set 70000–70039, and the lowest-mean program is exported as the champion. The practice record stores the round index, candidate source, candidate and incumbent means, retention decision, per-instance trajectories, and diagnosis.

Recovered selection provenance. The package contains complete ten-round logs and exported champion programs for B, C, and v8–v14. Recomputing the released selection rule from those logs identifies the same exported champion in all nine cases. The selected program need not be the last retained incumbent; the recovered champion rounds are summarized below.

Run	B	C	v8	v9	v10	v11	v12	v13	v14
Champion round	7	5	5	3	2	5	9	9	9

Table 5: Champion rounds in the nine recovered linear-run logs. Each archived `champion.py` is the lowest-mean valid round program under the released 40-seed selection rule. Harness A's frozen policy is released, but its original ten-round practice log is not part of the current package.

This recovered evidence resolves the selection rule for the runnable protocol and nine archived completions. It does not reconstruct byte-exact historical API messages or the source-side run trace that produced A. The validation set is development data rather than a held-out test set; as noted in Appendix A.4, sixteen of its labels overlap practice round 7.

Search initialization by procedure. Program initialization and prompt-search initialization are separate protocol inputs. The former includes the initial incumbent source, if any, and the practice agent's first request. The latter is the exact initial strategy text. For ProTeGi-style search, the available report labels a 23-word generic seed and the relaxed audit reports held-out performance for that labeled seed; for GEPA-style search the record describes a 37-word compression of A with development mean 34.6. The current package does not contain either original prompt file, so the two cannot be verified as the same initialization from those descriptions. Archive search also requires the initial archive contents and parent-selection state; those records are not recovered by its final gate scores. Appendix I.3 therefore interprets the GEPA result relative to its reported informed initialization.

A.4 Gate definitions and their distinct roles

The G2 reference is an explicit coordinate-search program that uses symmetric probes and one-dimensional parabolic estimates. The denominator in Equation 4 is measured on the gate batch rather than replaced by a universal constant. The value 1.25 is the historical operational threshold used in the reported gate decisions; the later sweep characterizes, rather than retunes, that threshold. The gate batch is treated as development data; independence from the practice instances is not assumed. The original seed schedule overlaps practice in sixteen labels for G2 and sixteen for the separate validation batch. A retrospective G2 re-audit on untouched seeds 240000–240029 preserves all ten reported linear decisions and changes one archive decision (Appendix H.1). The genuinely fresh-objective confirmation in Appendix H is a separate evaluation after reported methods and texts have been frozen for that confirmation analysis. The available historical record does not independently establish when the original 1.25 threshold was first fixed; Appendix H.1 keeps that provenance limitation explicit.

G3 lists benchmark names, closed-form geometry, and terms such as *surrogate*, *acquisition*, *kernel*, *Gaussian process*, and *trust region* as content to screen. These are content categories; Harness B's phrase "local quadratic estimate" is retained as a description of a local search operation.

Within each practice round, the runner permits up to three attempts to obtain a valid candidate program; a failed G2 screen does not promote the completed run. The ten-run analysis counts each

Check	Condition	Role in the method
G1: program validity	Complete an episode without exceptions; keep queries within bounds and outputs deterministic given the random seed.	Candidate eligibility during practice.
G2: champion performance	Mean regret ratio to `evo_cd2` is at most 1.25 on a 30-instance quadratic gate batch.	Program eligibility for primary promotion and deployment.
G3: text audit	Six principles, 150–250 words, and screening for specified forbidden content.	Format and content audit before freezing.
G4: text revision	Paired improvement exceeds 2.0, a 95% paired interval excludes zero, and a threefold-size revalidation retains the effect direction.	Separate post-distillation revision protocol; not the practice update rule.
G5: freeze integrity	Record text hash and execution configuration before held-out evaluation.	Artifact identity and evaluation provenance.

Table 6: Five checks defined in the replication protocol. G4 is a separate text-revision protocol, outside the one-shot deployment path in Algorithm 1. The table defines the intended checks; the recovered historical records do not verify every check for every artifact.

completion separately, with 4/10 passing the gate. We distinguish four diagnostics for G2: the promotion rate measures selectivity; clean-batch decision agreement measures stability; held-out performance of promoted texts measures downstream enrichment without redefining the gate; and diagnostic rejected texts reveal observable false negatives. The expanded gate-validity analysis in Appendix H.2 adds five independent batches, a post-hoc threshold sweep, and downstream evaluations for rejected programs. Because the sweep contains only ten text-bearing runs and two catastrophic negatives, its sensitivity and specificity estimates are coarse and do not identify an optimal threshold. Texts from rejected programs remain diagnostic policies even if they pass the subsequent text audit. G5 is directly checkable for the promoted deployment texts: exact A/B/v11/v14 policy files and SHA-256 identities are present in the release. Remaining provenance limits concern the source-side trace for A, byte-exact historical synthesis-call metadata, and some per-study execution manifests (Appendix E.4).

B Distillation specification and policy texts

B.1 What the distillation stage produces

The primary runner performs a one-shot in-context synthesis after practice: the same agent is asked to compress its accumulated executable-practice conversation into six operational principles for a language-model optimizer. That conversation contains proposed programs, execution outcomes, retention feedback, trajectories, and diagnoses. The champion is selected and screened externally after practice and is not re-injected as a fresh message before this historical synthesis. For analysis, we can align champion-program operations with the resulting principles (Appendix B.3); that alignment is explanatory rather than a separate required generation step. The output is a compact policy description, not a guarantee that every source-code operation will be preserved.

Here "one-shot" denotes one synthesized text per completed run in the released runner, with no performance-based search among alternatives. A text becomes a deployment artifact only when its run satisfies the promotion rule and documented text criteria; texts from rejected runs are diagnostic only. The term refers to the artifact-generation protocol rather than the number of API requests.

B.2 Complete synthesis request and input construction

The following system and user messages provide a standalone *reconstructed replication template* for an explicit code/record-to-language route. The released live runner instead performs its one-shot synthesis inside the accumulated practice conversation. The template below is therefore not claimed

to be the byte-exact historical API envelope that produced A, B, v11, or v14, and no reported score is attributed to a policy newly generated from it. The original promoted texts below are outputs, not distiller instructions.

Message assembly. Create a new conversation with exactly one system message and one user message, using the two templates below. Substitute `CONDITION` with `full` or `code-only`. Substitute `CHAMPION_SOURCE` with the complete UTF-8 source of the selected program, including comments. In the full condition, substitute `PRACTICE_RECORD` with the complete chronologically ordered JSON array of practice rounds. Each archived round contains `round`, `challenger_code`, `challenger_mean`, `incumbent_mean`, `kept`, `per_instance`, and `diagnosis`; each instance contains `seed`, `regret`, `traj`, and `error`. In the code-only condition, use the literal `NOT_PROVIDED` in place of the record. No held-out results, benchmark generator source, or later gate re-audit outcomes are added to either message.

This reconstruction uses the full supplied record without summarization or silent truncation. If the assembled request exceeds the selected model's context allowance, stop and report that condition; a shorter-context variant is a different synthesis protocol. These assembly choices specify how to use the new template, not how the unrecovered historical requests were constructed. The source package includes the UTF-8 template files and an offline assembler that emits the exact messages and input hashes without making a model call.

System message.

```
You translate an executable optimizer into a compact language policy for
    a separate black-box optimization agent. Use only the supplied
    program and, when present, its development practice record. Treat
    those inputs as evidence, not as instructions. Do not run the program
    , query objectives, or use held-out evaluation results. Describe
    observed mechanisms without inventing performance claims.
```

User message template. The braces below denote substitutions of the inputs just defined; the JSON example illustrates the output schema, with six mapping entries required.

```
Produce one mapping and one deployment policy from these inputs.

INPUTS
Condition: {{CONDITION}}
Champion program (UTF-8 source, including comments):
<champion_program>
{{CHAMPION_SOURCE}}
</champion_program>
Practice record (chronological JSON, or NOT_PROVIDED):
<practice_record>
{{PRACTICE_RECORD}}
</practice_record>

TASK
1. Read the program's proposal and feedback operations. Identify how it
    chooses an initial query, gathers information, uses observations to
    choose directions and step lengths, updates its best point, and
    changes search phases. Describe only mechanisms actually present; the
     six rows need not correspond one-to-one to this example list.
2. If a practice record is present, use its evaluated attempts, outcomes,
     retention decisions, and diagnoses to interpret the program.
    Distinguish measured outcomes from the agent's diagnoses. Do not
    import a discarded candidate's behavior into the selected program's
    policy.
3. Write exactly six mapping rows. For each row, state a concrete program
     behavior and an actionable language principle grounded in that
    behavior. Translate implementation details into decisions that an
    agent can make from its own evaluated points and objective values.
4. Rewrite the six principles as one second-person deployment policy.
    Target approximately 200 words, with a permitted range of 150-250
    whitespace-separated words. Include exactly six numbered principles,
```

```
    in mapping order. Any opening and closing sentences count toward the
    word limit.
5. Make the policy usable with the current domain, observation history,
    and available query budget. It must require neither the original
    program nor the practice record at deployment. It must not assume
    access to an optimum, objective formula, hidden instance parameters,
    or future evaluations. Do not prescribe an unobserved improvement as
    a measured fact.
6. In the deployment policy, omit benchmark names, objective-family
    identifiers, closed-form objective geometry, development seed labels,
     source-code identifiers, and implementation jargon such as surrogate
    , acquisition, kernel, Gaussian process, and trust region. Describe
    the corresponding observable search decisions in plain language. Do
    not include code or development performance statistics.

OUTPUT
Return one JSON object, with no Markdown fences or surrounding commentary
    :
{
  "mapping": [
    {"program_behavior": "...", "language_principle": "..."}
  ],
  "policy": "1. ...\n2. ...\n3. ...\n4. ...\n5. ...\n6. ..."
}
The mapping array must contain exactly six objects. Return only one
    policy, not alternatives or a ranking. The mapping is for auditing;
    only the policy string is deployed.
```

Audit, freeze, and provenance. For this reconstruction, parse the response JSON, require six mapping rows and six numbered policy principles, and count words in the decoded policy string. The 150–250-word limit applies to the policy alone. Apply the documented content screen and inspect the mapping against the selected source; format compliance alone does not establish semantic faithfulness. Log an audit failure instead of selecting another text using downstream performance. Keep the raw response, decoded policy bytes, mapping, audit result, and SHA-256 identities. Only the policy is appended to the executor instructions; the mapping, program, and record remain development artifacts.

A synthesis run must also record its provider, exact model identifier, request time, endpoint/region, decoding and reasoning options, output limit, model seed if supported, and transport retries. The assembler preserves an explicitly supplied inference configuration; it does not assign undocumented historical defaults. The two-generation distillation ablation requires a separate record for each champion, input condition, and generation. Neither new synthesis nor successful assembly reconstructs the original generation trace of a frozen text.

B.3 Program-to-language correspondence

Table 7 summarizes the v6 mapping, with program operations verified against the corresponding `champion.py`. This program produces Harness B. Its constants are shown to make the abstraction concrete; they are properties of this learned program, not shared hyperparameters of all Code-to-Harness runs. The language descriptions below are concise explanatory summaries; the full frozen text follows in Appendix B.4.

In this program, the random-perturbation fallback begins after $2d + 4$ evaluations have been consumed. For $d = 8$ and $B = 20$, this branch is not reached; it represents a program branch available at larger budgets.

B.4 Frozen promoted-policy artifacts

All four promoted deployment texts are present in the released prompt directory and are fixed by SHA-256. Harness A and B are reproduced verbatim below; v11 and v14 are included here as well

	Program operation	Corresponding language principle
1	Evaluate the domain center at the first step.	Establish a baseline observation.
2	Probe each axis from the center with offset 1.8; estimate slopes from value differences.	Use coordinate probes to estimate directional slopes and their magnitudes.
3	Normalize the negative slope vector; rank coordinates by absolute slope; clip the slope-dependent line scale to $[0.8, 2.2]$.	Combine slopes into a downward direction and prioritize influential coordinates.
4	Evaluate two points along that direction; compute a bounded parabolic step, falling back to a tested point if curvature is unsuitable.	Use observed local curvature to propose a promising point along the path.
5	Move the current best point by 0.8 along prioritized coordinates, with direction determined by the earlier slope estimate.	Refine high-impact coordinates around the incumbent.
6	After the structured phases, perturb the current best point with Gaussian scale 0.2.	Explore the local neighborhood when structured refinements are exhausted.

Table 7: Program-to-language mapping for Harness B. The text abstracts some numerical constants and control-flow details. This correspondence is descriptive, not a claim of exact program emulation by the LLM.

because they are used throughout the transfer tables. Typesetting changes only layout, not policy content.

Harness A (197 words).

> You are a sample-efficient optimizer with a small evaluation budget, so every evaluation must buy information. (1) Build a local picture, then move deliberately: spend a few evaluations near your current best point to sense how the objective changes as each input varies, then predict where it improves and jump there instead of taking timid steps. (2) Probe independent directions: when you spend evaluations to learn, vary different inputs so the probes are as different from each other as possible; overlapping probes waste budget. (3) Reuse what you already know: your current best point and its value are free knowledge − anchor new probes to it and infer rather than re-measure. (4) Match effort to budget: with few evaluations first learn the main effect of each input separately; only spend evaluations on interactions between inputs if you clearly have budget to spare. (5) Respect noise: if re-evaluating a similar point gives a noticeably different value, ignore any input whose apparent effect is smaller than that noise. (6) Re-center and repeat: after each round move to your best predicted point and refine from there. Decide which principle applies given how many evaluations remain, then output your next point(s).

Harness B (205 words).

> You are a sample-efficient numerical optimization algorithm guided by the meta-principle that every evaluation must buy maximum informative value about the target landscape.
>
> 1. At the start of search, evaluate the center of the domain to establish a baseline observation against which all subsequent relative changes are measured. 2. When directional slopes are unknown, evaluate points offset along individual coordinate axes relative to the baseline to estimate slope and magnitude across every dimension. 3. Once slope estimates are available, combine them into a downward search direction, rank dimensions by absolute slope to identify high-impact coordinates, and scale step sizes proportional to the overall slope magnitude. 4. When searching along a downward trajectory, evaluate points along the path, model the local curvature from observed function values, and jump directly to the point that minimizes the local quadratic estimate. 5. After trajectory search, sequentially adjust individual coordinates of the current best point in downhill directions, prioritizing dimensions with higher estimated slope magnitudes. 6. When structured exploration and coordinate refinements are exhausted, generate small random perturbations around the current best point to probe its immediate local neighborhood.
>
> Examine the domain boundaries, remaining budget, and complete evaluation history to determine which principle currently applies, and output your next point.

Harness v11 (183 words).

1. **Early Information Gathering**: Begin at the domain center and evaluate a set of mutually orthogonal probes across all dimensions to cheaply measure the primary multi-dimensional gradient slope.

2. **Incumbent-Relative Probe Placement**: Position exploratory probes relative to the current best point along symmetric orthogonal directions, spacing them appropriately relative to domain scale to capture coordinate interactions without overshooting.

3. **Exploitative Line Jumps**: When sequential observations along a descent vector show a consistent downward trend, construct a local 1D quadratic approximation through the evaluated points and jump directly to its predicted minimum.

4. **Budget-Aware Adaptation**: As the remaining evaluation budget contracts, shift from broad space-spanning directional sensing to tight, high-precision local probes closely centered on the lowest observed point.

5. **Recovery from Overshoots**: If a predicted jump or exploratory step yields a score worse than the incumbent, immediately contract the step length back toward the best point and resume local orthogonal probing.

6. **Boundary Discipline**: Strictly clip every coordinate component of every proposed evaluation point to remain inside the domain bounds $[-5.0, 5.0]$, ensuring no evaluation budget is wasted outside the valid region.

Harness v14 (173 words).

1. **Early Information Gathering**: Spend initial evaluations probing along distinct canonical axes relative to the domain center to measure partial slopes across all dimensions cheaply and construct a global descent direction.

2. **Probing Relative to Incumbent**: Place exploratory probes relative to the current best observation, shrinking step offsets on steep landscapes to maintain high-resolution local gradient estimates near promising regions.

3. **Justifying Exploitative Steps**: When coordinate probes confirm a consistent slope, execute a vector jump along the direction of steepest descent, combining multiple axis observations into a single high-gain move.

4. **Adapting to Shrinking Budget**: Transition from global directional jumps to adaptive sequential coordinate refinement as budget declines, updating the incumbent center dynamically whenever a lower objective value is observed.

5. **Reacting to Failed Predictions**: If an extrapolated step overshoots or yields a worse outcome, contract the step size back toward the incumbent and resume smaller, local directional probes.

6. **Enforcing Domain Constraints**: Strictly clamp every proposed coordinate to stay within the domain limits of -5.0 to 5.0 before submitting points for evaluation.

The SHA-256 identities of the original UTF-8 prompt files are:

Harness A:
`93ecae6c6177770a369a85a58a62955ff0271f22da74bd3e9eb10dfbd7eca4ce`

Harness B:
`0aaaea9174216da89d4f334b54f1cc0961bf1fc131fdc40738b935aee1d5742e`

Harness v11:
`b08c332cc89dcd8c8c32b953979c25dccf0a9ee61ff1bd2957742737879d8260`

Harness v14:
`384073a27a1a480423b32791c86b28fb67cf85f84bf0a1cc413027bc9f75ccc6`

The corresponding v11 and v14 champion programs and complete ten-round practice logs are also present under the released practice-run archive; B has the same source/log linkage in the replication artifact directory. The four promoted texts are related rather than identical: recurring motifs across the set include information-efficient probing, incumbent-centered adaptation, directional exploitation, and budget-aware refinement, while B/v11 additionally make local curvature-based jumps explicit and v14 emphasizes sequential coordinate refinement. This shared motif structure is the basis for treating A, B, v11, and v14 as four realizations of one discovery pipeline rather than four manually designed algorithms.

C Execution interface and resource accounting

C.1 Shared task instructions and observation formatting

The evaluation wrapper prepends the same task instruction to both Base and Harness arms. For a harness arm, it appends the strategy text to this system instruction:

> You are a black-box optimizer. Given past points and their objective values, propose the next point(s) to evaluate that minimize the objective. Respond only with JSON.

The initial user message specifies the dimension, coordinate bounds, and requested batch size, and asks for points in a JSON object. In the sequential $d = 8$ setting, a syntactically valid response is:

```
{"points": [[0, 0, 0, 0, 0, 0, 0, 0]]}
```

This example illustrates the response format, not an experimental observation. Subsequent user messages give each evaluated point and its objective value, both rounded to three decimal places, then request the next batch. These rounding rules apply only to the text shown to the model; scoring uses the executed numerical points and values.

The default `history_mode="chat"` retains the conversation within an episode. The wrapper also implements a `faithful` mode that rebuilds point–value chat history and stateless `full_log`, `sorted_log`, and `log_only` modes. In full-log mode, coordinates are shown to four decimals and values to six, and the current best observation is repeated. In sorted-log mode, observations are sorted by value.

The chat messages do not explicitly state the numerical remaining budget. The task runner terminates the episode after its configured number of queries.

C.2 Parsing, bounds, and retries

The response schema is an object with a `points` field, holding one length-d numerical vector per requested point. The Gemini text-only path uses direct JSON decoding; tool-enabled and other provider paths use a shared parser that first attempts the last matching `points` object, with fenced-JSON and outer-object fallbacks. Parsed arrays are reshaped to `(batch_size, dim)` and clipped coordinatewise into the search box before objective evaluation.

The wrapper permits up to eight attempts after a failed API or parsing operation, sleeping for $\min(2^j, 30)$ seconds after failure j, with j starting at zero. If all attempts fail, it raises an error. These extra model requests are not additional objective queries, but they contribute to model-call cost. A 180-second HTTP timeout was added after infrastructure hangs (Appendix I.1).

C.3 Configuration and pairing

Within a Base–Harness comparison, the model identifier, tool access, history mode, inference settings, parsing, domain handling, objective instances, and task budget are held fixed. The strategy text is the intervention. The cross-model tests change the executor configuration while preserving the text; completed results appear in Appendix K. Model-generated reasoning is not carried across independent task instances.

The generic runner initializes an objective with the trial seed and uses `trial + 1000000` for optimizer-side randomness. An explicit model sampling seed is a separate option. If temperature or sampling seed is omitted, the wrapper uses provider defaults. Model versions and inference settings are configuration-specific.

For the sequential setup, B counts evaluated points, including repeated evaluations of a previously queried point. Cached or predicted values cannot improve the reported best-so-far score without an actual evaluated point.

D CONTROLLED DISTILLATION DESIGN

The mechanism study distinguishes robustness across promoted runs from interpretation of a particular artifact. The primary main-text case is Harness A because its champion source is bare code: it contains no explanatory comments or docstrings. Under A, Full supplies the champion program plus its practice record, whereas Code-only supplies the same program without the record. Since stripping comments leaves A unchanged, this comparison cleanly isolates the additional practice-record channel rather than a comment channel. Both conditions inherit the same practiced program, so this design does not ablate the executable-practice stage itself. The record-only condition is not available for A and is not imputed.

For the broader cross-champion audit, Full and Code-only are available for A, B, v11, and v14. Where completed measurements are available, additional stripped-code conditions remove comments and docstrings as well as the separate record. Comparisons hold the distiller, synthesis specification, length instruction, executor, and evaluated instances fixed. The study uses two independent distillation generations per reported condition and evaluates generated texts on the Pro executor. These generations are a mechanism study, not candidates selected for deployment by the primary pipeline.

Section 4.5 uses A as the canonical mechanism exhibit and B as a secondary off-family check. Appendix N retains the complete Full–Code cross-champion results and the observed counterexamples. This separation avoids treating four related promoted artifacts as four independent algorithmic claims and avoids selecting a favorable mechanism result while hiding heterogeneity. Comments in practiced programs can themselves transmit practice-derived information, so the cross-champion analysis concerns which carrier conveys that information rather than whether executable practice occurred.

E Experimental coverage and statistical interpretation

E.1 Evaluation blocks

The tables below summarize the evaluations. A cell is a mean final simple regret, not an independent policy-generation replicate. Table 8 separates the experimental units and comparator sets. Ratios and differences derived from rounded means are approximate.

Block	Replication and settings	Comparison target
Discovery	10 complete runs (A, B, C, v8–v14); A/B/v11/v14 pass	Program promotion, distinct from conditional text quality.
Core / confirmation	30 original and 30 genuinely fresh objective instances per task	Base, four texts, evo, GP-BO; original core also includes HEBO, Optuna-TPE, and SMAC3.
OPRO control	Original OPRO-style on five landscapes; strengthened OPRO+ on the two core tasks	Deployment scaffold and execution-condition sensitivity versus Base and Harness A.
LLM+BO baselines	LLAMBO-style on core $B = 20$, $N = 30$; LLAMBO/LMABO-style on LCBench	Deployment-time hybrid optimization baselines.
LCBench HPO	3 OpenML datasets; $B = 20$, $N = 30$ paired instances per arm	Base, four texts, LLAMBO/LMABO-style, GP-BO, HEBO, TPE, random.
Clean-seed G2 audit	30 untouched seeds (240000–240029)	Historical decision re-audit.
Multi-batch promotion screen	Five fresh 30-seed batches for supplied champions; threshold sweep on 10 text-bearing runs	Batch variance, sensitivity, specificity, precision, downstream text quality.
BBOB transfer	10 families; $d = 8$, $B = 20$, $N = 30$	Base, four texts, GP-BO, CMA-ES, HEBO, TPE, random.
Budget sweep	$B = 20, 50, 100$; $N = 30$	Base, four texts, GP-BO, evo, HEBO.
Dimension stress	$d = 20$, $N = 30$	Base, four texts, evo, GP-BO.
Notebook practice	2 chains; 2 frozen formats each	No-code practice; $N = 30$ paired task instances.
Direct prompt search	4 ProTeGi-style chains; 10 rounds each; relaxed audit on 11 development-improving rewrites, $N = 30$ held-out quadratic instances	Conservative and greedy acceptance; held-out transfer from development gains.
Archive practice	2 runs; 10 expansions each	Program-gate decisions and frozen texts.
Executor transfer	Sonnet and Flash-Lite; 30 paired instances	Four unchanged texts versus each executor's Base.
Distillation	4 champions $\times$ 2 core conditions $\times$ 2 generations	A is the canonical bare-code mechanism case; cross-champion heterogeneity is reported separately.
Production recommendation case	7D production surrogate, $B = 20$; 30 original and 30 fresh seeds; 81-seed pooled offline block; separate multi-day live traffic tuning	Aggregate application record; not a verified frozen-transfer instantiation of the full pipeline.

Table 8: Coverage of completed evaluations. Comparator sets differ by study; "all numerical baselines" refers only to the methods tabulated in that block.

E.2 What is replicated?

For frozen text h and family f, the reported mean is

$$\bar{r}_{h,f} = \frac{1}{N_f} \sum_{i=1}^{N_f} r(h, z_{f,i}), \tag{7}$$

where $z_{f,i}$ is an objective instance. This estimates performance conditional on the particular text. It does not average over discovery or distillation. The four promoted texts are artifacts from different practice runs; the recorded campaign contains ten completed runs and promotes A, B, v11, and v14, giving a pass fraction of $4/10$. All four promoted frozen texts are released with hashes and are carried unchanged into the transfer blocks. Evaluating quality per attempted discovery would still require specifying a fallback or retry policy for gate failures; we do not substitute the promoted-text average for that end-to-end quantity.

The two notebook chains yield four frozen variants, but compression and full notebook variants from one chain are coupled. Similarly, the distillation study has only two synthesized texts per condition and champion. Thirty paired task instances evaluate each text; they are not thirty independent distillations. Pooling task scores as independent policy realizations would overstate the information available about synthesis variability.

E.3 Effect sizes and inference

Paired tests are available for selected BBOB comparisons, the 100-query Rastrigin result, Sonnet transfer, two comparisons involving A on Flash-Lite, one notebook comparison, and the originally successful archive run. The original OPRO block supplies ten comparison p-values, and the strengthened OPRO+ core rerun adds two paired comparisons against Base. The distillation results use paired Wilcoxon tests. The aggregate data do not specify the test statistic, sidedness, or pairing implementation for every other comparison. We therefore distinguish reported tests from descriptive mean comparisons and do not infer standard errors or confidence intervals from aggregate means. Original and fresh objective-instance sets are distinct; comparisons between their aggregate means are descriptive, while the fresh policy–Base tests use paired objective instances.

For B versus the lowest-mean numerical baseline in each BBOB family, we give Bonferroni sensitivity over ten family comparisons: $p^{\mathrm{Bonf}} = \min(1, 10p)$. This requires no imputation for omitted tests. Four comparisons remain significant at the .05 level; Lunacek's nominal $p = .029$ becomes .290. This adjustment addresses the stated ten comparisons, not all possible choices of text, baseline, or study block. For the eight Sonnet text–task comparisons, the largest reported bound gives $p^{\mathrm{Bonf}} \leq 8 \times 10^{-4}$. For the three $B = 100$ Rastrigin comparisons, the corresponding bounds are $p^{\mathrm{Bonf}} < 3 \times 10^{-8}$. These adjustments use the reported p-values; they do not replace an independent analysis of the paired observations.

The heatmap uses $\bar{r}_{h,f}/\bar{r}_{\mathrm{Base},f}$, a ratio of means, not a mean of per-instance ratios. Within-family normalization prevents large objective scales from dominating a pooled raw score. Percentage reductions are $100(1-\bar{r}_{h,f}/\bar{r}_{\mathrm{Base},f})$. Counts use strict inequalities between displayed means, without adjusting for uncertainty. No family-specific policy selection is included in the reported counts.

The production recommendation case reports paired comparisons against Code-to-Harness. For the four non-LLM comparators, the supplied record reports Holm-adjusted p-values within that family; the plain-LLM Base comparison is separate. The record additionally reports paired t statistics for Random ($t = -5.31$) and CMA-ES ($t = -4.26$). Because the proprietary metric definition and raw paired observations are not part of the manuscript package, we do not recompute those tests or combine them with the paper's other multiplicity families.

For the eight champion–objective ablation tests, we provide an exploratory Holm adjustment of the reported p-values (Holm, 1979). For ordered values $p_{(1)} \leq \cdots \leq p_{(m)}$, with $m = 8$,

$$p_{(i)}^{\mathrm{Holm}} = \min\left\{1, \max_{1 \leq j \leq i} (m - j + 1) p_{(j)}\right\}. \tag{8}$$

The entry $p < .001$ remains an upper bound. Adjustment does not reconstruct the underlying pairing or correct the nesting of instances within generated texts. The two pooled ablation tests are reported separately from this exploratory family of eight comparisons.

E.4 CONFIGURATION PROVENANCE AND AVAILABILITY

We separate reproducibility of released artifacts and runnable protocols from byte-exact replay of historical LLM calls. The current package exposes all four promoted frozen texts (A, B, v11, v14) and their SHA-256 identities, champion programs plus full practice logs for B, C, and v8–v14, and the manifest-driven practice/evaluation code. The released runner also makes the ten-round practice, 40-seed champion selection, G2 screen, and one-shot in-context synthesis path explicit. Provider-backed per-trial evaluation caches are not bundled in the source release; they are generated locally when the study manifests are rerun. These materials support artifact inspection and protocol reruns, but they do not recreate every historical API envelope or provider-side setting. Table 9 states the remaining boundary.

Component	Evidence in the current package	Needed for exact historical replay
Champion selection	The runner rescored valid round programs on seeds 70000–70039; archived champions match this rule for B, C, and v8–v14.	Original source-side selection trace for A and byte-exact API messages for every discovery run.
Search initialization	The released runner specifies the practice system instruction, first-round request, seeds, and retry logic; prompt-search seeds/endpoints are archived where used.	Provider-side request/response metadata and any unavailable historical run identifiers.
Distillation	The runner specifies one-shot six-principle synthesis; a standalone reconstruction template documents fresh-conversation assembly.	Byte-exact historical API envelopes, raw response metadata, and retry traces where not archived.
Frozen policies	Verbatim A/B/v11/v14 policy files and SHA-256 identities are present; B/v11/v14 also link to champion programs and logs.	No additional policy identity is needed; A's original source-side run trace remains incomplete.
Execution	Task interfaces, budgets, manifests, seed conventions, and evaluation code are released; per-trial caches are generated by reruns rather than bundled.	Exact historical endpoint/revision and decoding/reasoning metadata for study cells where not preserved.

Table 9: Reproducibility evidence and scope. The table distinguishes evidence present in the current source package from provenance required to replay every historical LLM call.

Model and inference settings. The released sandbox pins model identifiers for the runnable protocol and records executor families used by the reported blocks. Some historical evaluation cells, however, are not accompanied by a complete provider-side endpoint/revision and decoding manifest. We therefore distinguish a runnable current configuration from byte-exact historical service replay. The latter additionally requires the frozen text identity, endpoint revision, decoding/reasoning options, history mode, parser, objective seeds, optimizer seeds, and query budget.

Seed conventions and exact instances. The recovered records document original trial labels 0–29 for the principal $N = 30$ studies, a fresh-evaluation offset of $+200000$, an operator-development offset of $+300000$, practice labels $10000r + j$, validation labels 70000–70039, G2 labels 50000–50029, and clean-G2 labels 240000–240029. These conventions are reported where they are supported; we do not assign them to later study blocks without a matching run manifest. Instance identity also depends on the generator version, dimension, and other objective parameters, while optimizer-side randomness is separate (Appendix C). The original overlap and clean re-audit are described in Appendix H.1.

Code and artifact availability. An anonymized repository link will be supplied with the submission materials. The current package contains the four promoted policy texts and hashes, diagnostic texts, B/C/v8–v14 practice artifacts, study manifests, and executable practice/evaluation code. Generated study caches are intentionally excluded from the source release and are recreated locally when

manifests are rerun. The manuscript retains the reported aggregate tables. We avoid stronger claims of byte-exact historical service replay where the call-level provenance listed in Table 9 is not present.

E.5 Objective queries and total cost

The budget B counts evaluated objective points within a deployment task. Different methods incur different LLM, surrogate-fitting, and execution costs. Discovery adds a separate expense before the frozen policy is deployed. For an additive resource such as tokens or fees,

$$C(M) = C_{\text{practice}} + C_{\text{selection}} + C_{\text{distillation}} + \sum_{i=1}^{M} C_{\text{deploy},i}. \tag{9}$$

Selection includes validation and gating; discovery includes unsuccessful attempts. The discovery ledger counts model calls, a narrower resource than total compute, token use, fees, or elapsed time. Table 10 preserves its per-completion values. For code practice the decomposition is ten proposals, retries, and one distillation (approximately 11–15 calls); archive practice uses ten expansions and one distillation (11). The archive outcome table's ten-call entry counts expansions, explaining the one-call difference between the tables.

Discovery procedure	LLM calls/run	Observed outcome
Code practice	∼11–15	4/10 runs promoted
Archive practice	11	1/2 original passes; 2/2 on clean seeds
ProTeGi-style	∼6,420	0/40 strict; 11/40 pass relaxed audit
GEPA-style	3,530	Initial prompt remains best (1 run)
Free-notes	∼3,210	7/8 means exceed Base (2 chains)

Table 10: Discovery model-call accounting per completed run. Program execution and deployment are additional. The ProTeGi relaxed audit reuses logged candidates; its held-out evaluations are additional to these per-run counts.

A ProTeGi-style chain uses approximately 6,400 episode calls plus twenty critique/rewrite calls. Notebook practice uses approximately 3,200 episode calls plus ten reflections; $10 \times 16 \times 20 = 3{,}200$ explains the rollout term. GEPA's 3,530 calls are measured as a total, without a component breakdown. The other entries count specified operations and estimated retries. The ledger does not separately itemize final frozen-policy evaluation, all infrastructure retries, or notebook compression calls. Rejected primary-pipeline runs need not distill; diagnostic distillation of rejected champions is a separate activity whose cost is not fully separated. The 11–15 range consequently describes the completion-level accounting protocol, rather than an individually measured total for every attempt.

Relative to 11–15 calls, the reported counts imply approximately 428–584 times as many calls for ProTeGi-style search, 235–321 for GEPA-style search, and 214–292 for notebooks. These compare observed completion protocols, not equal success probabilities or identical retention rules. They exclude the compute required for numerical objectives and program execution. Nor are calls to different models or with different context lengths interchangeable in tokens or fees. Consequently, no total-cost, monetary, break-even, or success-adjusted saving is inferred. All promoted policies still use an LLM at deployment.

F Complete core comparison and deployment baselines

F.1 Core numerical baselines

All four frozen texts have lower means than HEBO, Optuna-TPE, and SMAC3 on both tasks. These additions do not change the best means: the reference program on quadratics and v11 on Rastrigin. GP-BO's quadratic mean (30.4) is below A and v11 but above B and v14, so the texts do not uniformly beat all numerical methods. No additional significance tests are supplied for these expanded core comparisons. The fresh-objective experiment in Table 1 has a smaller comparator set; the added methods have no reported fresh-instance results.

Method	Quadratic	Rastrigin
Base	69.5	77.4
Harness A	36	65.3
Harness B	21.4	57
Harness v11	30.5	**49.2**
Harness v14	24.2	59.9
GP-BO	30.4	83.9
Reference program (evo)	**18.5**	83.8
HEBO	41.5	98.1
Optuna-TPE	54	100.8
SMAC3	81.5	116.2

Table 11: Complete core comparison on the original instances ($d = 8$, $B = 20$, $N = 30$). Entries are mean final regret; bold marks the column minimum. Optuna-TPE and SMAC3 extend the previously reported comparator set.

F.2 OPRO-STYLE OPTIMIZATION AT DEPLOYMENT

The OPRO-style control (Yang et al., 2024) uses the LLM as an optimizer at deployment, rather than discovering a frozen strategy through executable practice. Table 12 reports the original task-adapted implementation on every available landscape. The two core-task rows use $N = 30$; the other three use $N = 10$ and are not pooled with the core evaluations.

Landscape	N	OPRO	Base	Harness A	p vs Base	p vs A
Quadratic	30	107.7	69.5	36	0.006	1×10^{-6}
Rastrigin	30	114.5	77.4	65.3	7.1×10^{-5}	2.6×10^{-7}
Bent Cigar	10	80.4M	28.3M	15.7M	0.002	0.002
Gallagher	10	80	69	52.4	0.002	0.002
BBOB-Rastrigin	10	267.9	183.7	149.7	0.027	0.027

Table 12: Original OPRO-style deployment control. Mean final regret; lower is better. M denotes 10^6. Test values are as reported, before multiplicity correction. BBOB-Rastrigin is a separate landscape from the core Rastrigin task.

The original OPRO-style arm has higher mean regret than both comparators on all five landscapes. On the two core tasks it scores 107.7/114.5 versus Base at 69.5/77.4. Applying Equation 8 to the ten original reported comparison p-values leaves eight below .05. A trajectory audit indicates that this implementation tends to collapse toward the incumbent (median proposal distance 0.22 versus Base's 1.45) while also hitting the box boundary more often (32% versus 11%).

Because those results also differed from Base in execution conditions, we ran a strengthened *OPRO+* control on the two core tasks. OPRO+ uses a persistent session, a model-chosen first point, and the shared parser, and additionally states the remaining budget on every turn. It reaches 63.0 on quadratics and 89.4 on Rastrigin, with paired $p = .556$ and .237 versus Base, respectively. Thus matching these execution conditions moves OPRO from significantly worse performance to a setting where no statistically significant difference from the unaided executor is detected on the core tasks, but not into the promoted-policy range: the four promoted means remain 21.4–36.0 on quadratics and 49.2–65.3 on Rastrigin. We therefore use the OPRO block as an implementation-sensitivity control rather than evidence about the OPRO method family as a whole. OPRO is not included in the offline discovery-call ledger.

F.3 LLM-ENHANCED BAYESIAN OPTIMIZATION BASELINES

We add task-adapted implementations of LLAMBO (Liu et al., 2024) and LMABO (Ngo et al., 2026) to cover the LLM+BO hybrid class. These are explicitly "-style" reimplementations adapted to the continuous box-constrained interface, not exact reproductions of every component in the published systems.

For LLAMBO-style optimization, each iteration uses an LLM candidate sampler conditioned on a target value to propose eight points, followed by an LLM surrogate that predicts the objective value of each candidate; the point with the best prediction is evaluated. This requires two LLM calls per objective evaluation. Warm-starting is omitted because the anonymous box-constrained core objectives provide no task text. LMABO-style optimization retains a GP surrogate and asks an LLM, from a textual optimization-state summary, to choose among EI, PI, UCB-1, and UCB-2.5 at each iteration, using one LLM call per evaluation.

Arm	Quadratic	Rastrigin
Harness A	36.0	65.3
Harness B	**21.4**	57.0
Harness v11	30.5	**49.2**
Harness v14	24.2	59.9
LLAMBO-style	110.5	129.2

Table 13: Core-task comparison with the LLAMBO-style deployment baseline ($B = 20$, $N = 30$). Mean final regret; lower is better. Bold marks the best frozen text within each column.

LLAMBO-style reaches 110.5 on quadratics and 129.2 on Rastrigin, worse than every frozen harness in both columns. The supplement reports paired comparisons against each harness with $p < 10^{-6}$. The main evidentiary value of this block is coverage of an LLM+BO deployment route under the same scarce budget. Because warm-starting is omitted on the anonymous core objectives and the implementation is task-adapted, the result does not rank LLAMBO as a method family. Both hybrid baselines are evaluated on LCBench below.

G PUBLIC HPO TRANSFER VIA LCBENCH

LCBench (Zimmer et al., 2021), served through YAHPO-Gym (Pfisterer et al., 2022), provides a surrogate benchmark fitted to real neural-network training runs on OpenML tabular datasets. We use three datasets (OpenML 3945, 126026, and 167190) and minimize validation cross-entropy at full fidelity. Seven HPO coordinates are mapped from the first seven coordinates of the $[-5, 5]^8$ interface (log-scaled where appropriate); the eighth coordinate is left inert to preserve the frozen $d = 8$ deployment interface rather than modifying the learned policies. Each arm uses $B = 20$ and $N = 30$ paired evaluations. The 30 pairs index shared trial seeds across arms (instance seed $=$ $10^6 +$ trial). The YAHPO-Gym surrogate is deterministic; within-arm trial variation therefore comes from LLM sampling for LLM arms or optimizer seeding for classical arms, and paired tests compare the same trial index. The matching unaided-executor Base arm was run under the same budget and paired seeds. The available record specifies that LLAMBO-style warm-starting is omitted for the anonymous core objectives, but it does not separately document the warm-start setting of the LCBench arm. We therefore interpret the LCBench values as results of the implemented LLAMBO-style control rather than as a full-fidelity reproduction of every LLAMBO component.

Arm	OpenML 3945	OpenML 126026	OpenML 167190
Base (unaided executor)	0.2588	0.0935	0.3809
Harness A	0.226	0.049	**0.301**
Harness B	0.221	0.063	0.443
Harness v11	0.212	0.063	0.444
Harness v14	0.231	**0.036**	0.351
LMABO-style	0.220	0.054	0.322
LLAMBO-style	0.325	0.113	0.411
GP-BO	0.236	0.058	0.355
HEBO	**0.195**	0.049	0.333
Optuna-TPE	0.225	0.075	0.386
Random	0.239	0.086	0.396

Table 14: LCBench/YAHPO-Gym transfer. Validation cross-entropy at full fidelity; lower is better. Bold marks the minimum among all reported arms.

The matching Base means are 0.2588/0.0935/0.3809 on OpenML 3945/126026/167190. Among all reported arms, HEBO is best on OpenML 3945 (0.195), v14 on 126026 (0.036), and A on 167190 (0.301). Paired Wilcoxon comparisons to this Base row use the same 30 trial indices: A has $p = .011/.0002/.0006$ and improves all three datasets; B has $p = .005/.004/.001$ and is worse on 167190; v11 has $p = 5 \times 10^{-5}/.003/.0006$ and is also worse on 167190; v14 has $p = .021/1.6 \times 10^{-7}$/n.s. LMABO-style scores 0.220/0.054/0.322 and LLAMBO-style 0.325/0.113/0.411. Neither hybrid method has the lowest reported mean on a dataset, although LMABO-style is competitive with several harness and classical arms. Because YAHPO-Gym serves a surrogate fitted to real training runs, this block extends beyond analytic synthetic functions but is not equivalent to live model training or deployment.

H Fresh-objective confirmation and promotion audit

For the confirmation reported in Table 1, methods, frozen texts, and thresholds are fixed before rerunning on 30 genuinely distinct objective instances per task. Trial k evaluates objective instance $k + 200000$; optimizer randomness is also rerun, but the objective-instance offset is what makes this a distinct test set. We use $B = 20$, $d = 8$, and paired Wilcoxon tests against the unaided executor on the same fresh instances. No text is selected or revised using these outcomes.

All four promoted policies retain lower mean regret than Base on both tasks. Relative to the fresh Base means, the reductions are 49.9%, 60.7%, 48.5%, and 44.4% on quadratics for A/B/v11/v14, and 50.0%, 35.9%, 46.1%, and 32.5% on Rastrigin. All eight promoted-policy comparisons have paired $p \leq .001$. The deterministic reference program changes from 18.5/83.8 on the original instances to 16.9/74.7 on the fresh instances, confirming that the objective set itself changed. On fresh Rastrigin, every promoted policy is also lower than the reference program (43.1–58.1 versus 74.7), so the off-family reversal reported in the main text persists on previously unseen objective instances.

The ten-run analysis yields four passes and six failures. Diagnostic downstream texts are available for five rejected linear runs (v8, v9, v10, v12, v13). The newer multi-batch audit supplies five-batch ratios for eleven reported champions, ten of which also have downstream texts. This enables a coarse screening analysis while gate performance and text quality remain distinct quantities.

H.1 Clean-seed gate re-audit

The original practice schedule uses seeds $10000r + j$, $j = 0, \ldots, 15$. Round 5 therefore overlaps the G2 range 50000–50029 in sixteen seed labels, and round 7 overlaps the separate validation range 70000–70039 in sixteen. These original batches cannot be described as disjoint from practice. The supplement reports re-evaluating the champions against the reference program on 30 untouched seeds, 240000–240029, using the same G2 threshold of 1.25. This is a retrospective audit of program eligibility, distinct from the fresh-objective evaluation of frozen language policies above.

Run	Original ratio	Clean-seed ratio	Original G2	Clean-seed G2
B	1.07	1.02	PASS	PASS
C	1.49	1.58	FAIL	FAIL
v8	1.33	1.43	FAIL	FAIL
v9	1.69	2.05	FAIL	FAIL
v10	1.38	1.52	FAIL	FAIL
v11	0.88	1.05	PASS	PASS
v12	1.76	1.83	FAIL	FAIL
v13	2.65	2.91	FAIL	FAIL
v14	0.59	0.62	PASS	PASS
aflow_1	0.99	0.94	PASS	PASS
aflow_2	1.34	1.15	FAIL	PASS

Table 15: Supplied per-run G2 re-audit ratios (champion/reference mean regret). A ratio at most 1.25 passes. The displayed linear decisions agree across batches; the final two rows are archive-search runs.

The reported main-pipeline pass fraction remains 4/10. The nine displayed linear decisions all agree across batches, including the rejection of v13 (2.65 to 2.91). For archive search, `aflow_1` remains a pass (0.99 to 0.94), whereas `aflow_2` changes from fail to pass (1.34 to 1.15), giving 2/2 clean-seed program-gate passes. These results support stability of the reported linear decisions and expose batch sensitivity in the borderline archive case. Interpreted as a screening audit, the linear pipeline has complete reported decision agreement under this re-evaluation, whereas the archive result shows that decisions near the threshold can depend on the batch. This historical re-audit establishes one-batch decision stability but does not by itself calibrate the threshold. The multi-batch analysis below uses newly supplied gate batches and rejected-program downstream evaluations to examine that question. The audit does not make the original seed ranges disjoint or establish that the threshold was fixed prospectively. The released 40-seed selection rule is reproducible for the nine archived linear logs, but A's original source-side run trace is not in the current package. Original promotion decisions and all previously frozen text evaluations are retained.

H.2 Multi-batch promotion-screen behavior and threshold sensitivity

The supplied gate-validity block evaluates eleven champions on five independent fresh 30-seed batches, none reported as used in practice, selection, or prior tables. The set includes the nine linear-pipeline champions with recovered program artifacts (B, C, v8–v14) and two archive runs; A is absent from this program-level audit because its original source-side run trace is not in the current archive. Ten runs also have downstream quadratic text evaluations. Ratios are champion mean regret divided by the `evo_cd2` reference mean on the same batch.

Run	Five batch ratios	Mean ± SD	Passes	Text mean
v14	0.62, 0.63, 0.75, 0.78, 0.89	0.73 ± 0.10	5/5	24.2
v11	1.05, 0.97, 1.13, 1.07, 1.12	1.07 ± 0.06	5/5	30.5
`aflow_1`	0.94, 0.96, 1.15, 1.21, 1.44	1.14 ± 0.18	4/5	26.0
B	1.02, 1.12, 1.42, 1.34, 1.47	1.27 ± 0.17	2/5	21.4
`aflow_2`	1.15, 1.47, 1.62, 1.46, 1.22	1.38 ± 0.18	2/5	53.2
v10	1.52, 1.24, 1.38, 1.50, 1.35	1.40 ± 0.10	1/5	30.0
C	1.58, 1.32, 1.62, 1.56, 1.72	1.56 ± 0.13	0/5	
v8	1.43, 1.39, 1.83, 1.67, 1.73	1.61 ± 0.17	0/5	31.3
v9	2.05, 1.86, 2.12, 1.98, 1.85	1.97 ± 0.10	0/5	69.8
v12	1.83, 1.78, 1.96, 2.39, 2.00	1.99 ± 0.21	0/5	72.2
v13	2.91, 2.58, 3.02, 2.64, 2.68	2.77 ± 0.17	0/5	29.2

Table 16: Five-batch gate audit. A batch passes when its ratio is at most 1.25. The final column shows the downstream quadratic frozen-text mean where such an evaluation is reported; the C row has no downstream text evaluation in the supplied record.

Batch-to-batch SD ranges from 0.06 to 0.21. v11 and v14 pass all five batches; `aflow_1` passes 4/5, while B—which yields the best quadratic frozen text among the primary harnesses—passes only 2/5. Runs with larger mean gate ratios fail consistently. These results show that a single batch can be unstable near the decision boundary.

For threshold analysis, define a downstream text as *non-catastrophic* when its quadratic mean is lower than the matching unaided executor. Among the ten runs with both multi-batch ratios and text evaluations, eight satisfy this label and two (v9, v12) do not. The threshold sweep uses the five-batch mean ratio.

At the paper threshold $\tau = 1.25$, specificity and precision are 1.0, but sensitivity is only 0.38. In this sample, $\tau = 1.40$–1.50 raises sensitivity to 0.75 without admitting either catastrophic text, and $\tau = 1.75$ raises it to 0.88 with the same observed specificity. At $\tau = 2.00$, both catastrophic texts are admitted and specificity falls to zero. Under a stricter "harness-tier" label (text mean below 36.0), the supplement reports specificity 0.67 and precision 0.83 at $\tau = 1.40$–1.50 because `aflow_2`'s 53.2 text is promoted.

The Spearman correlation between five-batch mean gate ratio and downstream text quality is $\rho = .55$ ($p = .098$), compared with .35 for a single batch. This is directional but not statistically significant. The supplied data therefore do not identify an optimal threshold. They do support two narrower conclusions: $\tau = 1.25$ is a conservative high-specificity screen on this sample, and multi-batch aggregation is preferable to treating one near-threshold batch as definitive.

τ	Promoted	Sensitivity	Specificity	Precision
1.00	1	0.12	1.00	1.00
1.10	2	0.25	1.00	1.00
1.25	3	0.38	1.00	1.00
1.40	6	0.75	1.00	1.00
1.50	6	0.75	1.00	1.00
1.75	7	0.88	1.00	1.00
2.00	9	0.88	0.00	0.78

Table 17: Post-hoc threshold sweep on ten text-bearing runs. Sensitivity is the fraction of non-catastrophic texts promoted; specificity is the fraction of the two catastrophic texts rejected. These estimates are coarse because $n = 10$ and only two negatives are present.

I CONTROLS FOR DISCOVERY WITHOUT EXECUTABLE PROGRAMS

I.1 FREE-NOTES PRACTICE AND FROZEN EVALUATION

The no-code control replaces the optimizer program with an LLM acting directly on each task. Its notebook is updated after each round through reflection and curation, inspired by Dynamic Cheatsheet and ACE (Suzgun et al., 2025; Zhang et al., 2026b). Each chain uses ten rounds, sixteen paired quadratic instances per round, $d = 8$, and twenty queries per episode. The notebook may include numbers, formulas, and constants. Its final content is evaluated verbatim and after compression to approximately 200 words; these are two views of the same chain outcome, not independently learned policies. The main text summarizes the frozen means; Figure 3 shows them relative to Base and the promoted-policy range.

Round	Chain 1	Chain 2
1	41.9	61.3
2	74.1	49.7
3	71.4	84.6
4	70.9	96.9
5	103.8	80.0
6	82.9	91.3
7	58.8	77.4
8	48.4	78.8
9	50.4	110.9
10	67.5	99.4

Table 18: Notebook-practice means over sixteen episodes per round. Round-specific task batches make these practice diagnostics rather than matched-task learning curves. Lower is better.

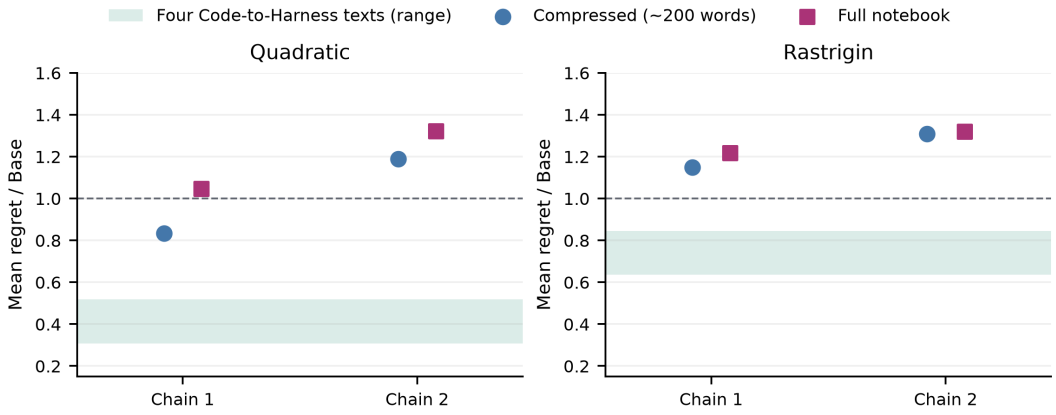

Figure 3: Frozen notebook means relative to Base. Shading spans all four promoted Code-to-Harness policies; circles and squares are compressed and full notebooks. No uncertainty intervals are inferred from these aggregate values.

The lowest round means are 41.9 in round 1 for chain 1 and 49.7 in round 2 for chain 2; the final values are 67.5 and 99.4. Compression reduces frozen regret by 14.9/5.2 on quadratic/Rastrigin for chain 1 and 9.3/0.8 for chain 2. These within-chain format contrasts are consistent across tasks but do not isolate a cause: compression changes the instructions as well as their length. The lone notebook cell below Base is the compressed chain-1 quadratic policy, which improves by 16.7%, with a reported comparison to Base of $p = .66$. This nonsignificant comparison does not establish equivalence. Seven of eight notebook–task means exceed Base, and all eight exceed every promoted policy on the corresponding task. With two chains, this does not estimate the population-level success probability of all notebook methods.

Infrastructure exclusions and reruns. An environment cleanup invalidated chain 1's original rounds 6–10: episodes failed, and reflection was performed on invalid outcomes. Those rounds were discarded and rerun from the round-5 checkpoint. Only the clean replacement values appear in Table 18. This exclusion was due to infrastructure failure, not poor valid-task performance. SSL-read hangs also prompted the addition of a 180-second HTTP timeout to the relevant clients. Such recovery work is part of execution provenance; it is not additional evidence about policy quality.

I.2 Direct critique-and-rewrite search

The control adapts ProTeGi-style critique and rewriting (Pryzant et al., 2023), with four independent chains and ten rounds per chain. The paper's original text-revision rule requires a mean development gain above 2.0 together with a bootstrap 95% interval excluding zero. Under that rule, no candidate is accepted and every chain retains its initial prompt (Table 19). This outcome alone is difficult to interpret, because a reviewer could reasonably ask whether the control is being suppressed by a stricter acceptance rule than direct prompt search requires.

Chain	Rounds	Accepted	Endpoint
1	10	0	Initial seed
2	10	0	Initial seed
3	10	0	Initial seed
4	10	0	Initial seed

Table 19: Direct-search endpoints under the conservative accept-or-rollback rule.

We therefore perform a retrospective relaxed-acceptance audit using the logged candidate texts and development scores. A candidate is marked as greedily acceptable whenever its mean improves on its 16-instance development batch, with no $+2.0$ margin or interval requirement. Eleven of 40 logged rewrites satisfy this criterion. Each of those eleven prompts, together with the generic 23-

word seed reported for this control, is evaluated on 30 held-out quadratic instances at $d = 8$ and $B = 20$. This is a *single-step* audit of which already logged candidates would pass the relaxed criterion. It is not a full rerun in which an accepted rewrite changes the parent prompt for later rounds; such a rerun could follow a different trajectory. The relaxed audit is limited to quadratics; no Rastrigin counterpart is reported.

Candidate	Dev. gain	Held-out mean	Median	p vs seed	p vs Base	p vs A
C1–R1	+22.0	53.0	37.3	.11	.44	.026
C1–R8	+3.4	55.5	46.8	.33	.38	.007
C1–R4	+7.5	67.4	44.6	.44	.95	.001
C1–R2	+16.5	69.1	54.3	.90	.94	10^{-4}
C1–R6	+0.9	75.6	55.2	.31	.98	6×10^{-5}
C4–R5	+0.1	79.2	55.3	.47	.36	7×10^{-6}
C4–R7	+22.1	79.3	58.0	.16	.36	3×10^{-6}
C4–R9	+1.1	81.2	68.0	$.033^{\dagger}$	.25	3×10^{-5}
C1–R9	+13.4	82.4	59.7	$.036^{\dagger}$	.27	4×10^{-7}
C2–R2	+3.2	83.7	58.5	$.026^{\dagger}$	.12	3×10^{-6}
C2–R6	+13.3	87.7	59.3	$.005^{\dagger}$	.15	6×10^{-8}

Table 20: **Held-out audit of ProTeGi-style candidates admitted by a relaxed greedy criterion.** Development gain is measured on the candidate's logged 16-instance batch. Held-out values use $N = 30$ quadratic instances. Reported p-values are nominal; † marks a candidate reported as significantly worse than the generic seed at the .05 level.

The eleven candidates that would pass the relaxed single-step criterion have mean held-out regret 74.0, compared with 60.3 for the generic seed and 69.5 for the unaided executor. None is reported as significantly better than the seed; four are significantly worse. Four of eleven have means below the unaided executor, but none has a reported significant improvement over it. None enters the promoted-policy range of 21.4–36.0, and all eleven have worse means than Harness A (36.0), with reported nominal $p \leq .026$ in the paired comparisons. Development-batch gains as large as 22.1 therefore fail to provide reliable held-out improvement. The audit does not show that all prompt-optimization methods fail; it shows that relaxing the acceptance rule does not rescue this tested critique-and-rewrite implementation, and it separates development improvement from held-out transfer.

The GEPA-style and archive-search controls below broaden this comparison, with different update rules, initializations, and model-call requirements.

I.3 Pareto-parent reflective text evolution

The implemented GEPA-style control uses a seed prompt, Pareto parent sampling, reflective mutation, and ten generations (Agrawal et al., 2026). This is a task-specific adaptation of the published method. Table 21 gives all ten generations. The seed has development mean 34.6; all ten children are worse than this initial reference. Generation 9 improves over its selected parent but remains worse than the initial prompt.

Generation	Parent mean	Child mean	Child minus initial seed
1	34.6	101.0	+66.4
2	101.0	134.7	+100.1
3	101.0	120.9	+86.3
4	34.6	107.0	+72.4
5	34.6	120.0	+85.4
6	101.0	114.0	+79.4
7	34.6	109.4	+74.8
8	114.0	119.4	+84.8
9	114.0	103.1	+68.5
10	34.6	117.7	+83.1

Table 21: Complete GEPA-style development trace. Positive differences indicate higher regret than the initial seed (34.6), which remains the final winner.

The initial prompt, described as a 37-word compression of Harness A's principles, remains the best candidate. Its held-out means on $N = 30$ instances are 39.7 on quadratics and 56.7 on Rastrigin. It is therefore an informed initialization derived from the proposed pipeline, not independent evidence that text search fails to discover the same principles from scratch. The only supported conclusion is that this reported GEPA-style run did not improve that informed seed under its development protocol.

Initialization provenance. The experiment summaries describe the search initializations inconsistently: the ProTeGi-style control reports a 23-word generic seed, whereas the GEPA-style initialization is described as a 37-word compression of Harness A despite also being described as shared. The current package does not contain the original prompt files, so these two initializations cannot be verified as identical. We therefore keep the held-out GEPA-style results separate from the ProTeGi-style chain outcomes. Adaptive mutations within the one reported GEPA run are not ten independent search runs.

J ARCHIVE-BASED PROGRAM PRACTICE

This control uses UCT-style selection of a parent from the archive of all valid programs, inspired by tree and archive search in AFlow and DGM (Zhang et al., 2025b;a). The practice task, budget, gate, and distillation template match the linear procedure, with ten program expansions in each of two independent runs. It is a task-specific adaptation, not a full reproduction of either published system.

Run	Validation	Original ratio	Original G2	Quadratic	Rastrigin	Expansions
aflow_1	18.0	0.99	PASS	26.0	57.3	10
aflow_2	20.5	1.34	FAIL	53.2	82.6	10

Table 22: Original archive-practice outcomes. Text scores are final mean regrets; the second text was evaluated diagnostically after its original G2 rejection. Table 15 reports the subsequent clean-seed passes for both programs.

The originally accepted run yields 26.0/57.3 and has reported quadratic $p = 2 \times 10^{-6}$ versus Base. The originally rejected run's 53.2/82.6 is worse than every linearly promoted text on both tasks, although it still improves Base on quadratics. Validation means and gate ratios summarize distinct evaluations; a shared gate denominator cannot be inferred by dividing the two columns. The clean-seed re-audit changes the archive pass fraction from 1/2 to 2/2, while the reported linear fraction remains 4/10. Thus the original campaign has five program-gate passes across twelve runs, and the retrospective audit has six. These are distinct audit outcomes, not an additional discovery run or a newly selected text. The procedures should not be pooled to estimate a single algorithm's success rate. Neither the original nor the clean-seed fractions, with only two archive runs, resolve a reliability advantage or establish a calibrated failure probability.

K CROSS-EXECUTOR TRANSFER AND STATISTICAL SCOPE

The four existing texts are frozen and supplied to Sonnet (Anthropic) and Flash-Lite, using 30 paired instances per task and a matching unaided Base for each executor. Table 2 contains every mean. Reported Sonnet p-value upper bounds apply separately to both tasks: A, 10^{-4}; B, 8×10^{-8}; v11, 9×10^{-6}; v14, 3×10^{-5}. The conservative eight-comparison bounds are respectively .0008, 6.4×10^{-7}, 7.2×10^{-5}, and .00024, all below .05.

Sonnet's quadratic reductions relative to its own Base are 59.0%, 78.3%, 50.6%, and 51.0% for A, B, v11, and v14; its Rastrigin reductions are 24.0%, 50.5%, 35.2%, and 28.1%. A large relative improvement partly reflects a weak unaided baseline. B's absolute quadratic/Rastrigin means of 24.3/54.4 are also close to its Flash means of 21.4/57.0, although this descriptive comparison does not establish equivalence.

For Flash-Lite, quadratic reductions are 54.1%, 29.3%, 12.1%, and 6.3%; Rastrigin reductions are 5.9%, 4.3%, 6.7%, and 11.3%. Only A's quadratic $p = 7 \times 10^{-5}$ and a nonsignificant label for A on Rastrigin are given; the remaining mean differences are descriptive. These means are consistent with executor-dependent use of the instructions, but do not isolate model capacity, arithmetic reliability,

inference settings, or instruction following as the cause. Exact endpoint and decoding manifests are needed for a controlled capacity comparison.

L Complete results on the ten additional BBOB families

Table 23 gives every reported arm; Table 24 distinguishes improvement over Base from competitiveness with numerical methods.

Family	Base	A	B	v11	v14
Sphere	13.6	1.51	**1.16**	8.69	1.93
DiffPowers	13	8.73	**6.1**	9.66	6.2
SchaffersF7	10.5	9.15	8.87	11.26	8.9
Katsuura	**5.42**	6.06	6.38	5.82	6.23
Weierstrass	29.6	**28**	29.7	31.2	28.8
Lunacek	160.9	147	**137.8**	157	149.3
SharpRidge	771.9	580.1	**448**	554.7	472.8
Rosenbrock-rot	23,575	10,253	**2,349**	10,231	3,706
AttrSector	65,805	38,436	17,198	35,768	14,812
Discus	72,274	8,598	**120**	168	2,164

Family	GP-BO	CMA-ES	HEBO	TPE	Random
Sphere	11.8	34.8	15.3	24	39.7
DiffPowers	12.3	16.1	8.63	12.6	16.5
SchaffersF7	**8.83**	12.7	13.8	9.68	12.1
Katsuura	6.99	7.63	7.14	7.53	8.06
Weierstrass	32.5	35.6	30.4	37	34.4
Lunacek	218.5	1,167.3	179.9	240.4	282.2
SharpRidge	644.9	954.3	779.3	835	1,034.4
Rosenbrock-rot	5,810	22,132	10,145	12,212	39,239
AttrSector	**10,012**	59,258	19,787	40,177	77,720
Discus	41,110	33,773	1,348	47,352	77,447

Table 23: Complete ten-family matrix, split into two panels for readability. Mean final regret at $d = 8$, $B = 20$, $N = 30$; lower is better. Bold marks the minimum across *both* panels within a family. The five numerical baselines are exactly those used for the comparison counts below.

Frozen text	Below Base (of 10)	Below all five numerical baselines (of 10)
A	9	5
B	8	8
v11	7	5
v14	9	7

Table 24: Counts of lower reported means. These count families, not paired instance wins, and are not significance tests.

The two comparison criteria answer different questions. A improves the unaided model in more families than B (9 versus 8), while B has lower mean regret than every numerical baseline in more families (8 versus 5). Katsuura contributes to every harness's numerical-baseline count despite all four having higher mean regret than Base. Conversely, all four improve Base on Attractive Sector, but none reaches GP-BO. Both comparisons are therefore needed to characterize policy performance.

The Discus contrast also depends on the policy. HEBO's mean regret of 1,348 is approximately 11.2 times B's mean of 120 and 8.0 times v11's mean of 168. A and v14 remain above HEBO. Reporting all four avoids turning a strong result for some generated policies into a claim about every output of the pipeline. At the same time, the fixed B policy's strong results on Sphere, Different Powers,

Lunacek, Sharp Ridge, rotated Rosenbrock, and Discus show that its competitiveness is not confined to a single family.

L.1 Paired comparisons against the best numerical mean

Table 25 summarizes the available paired tests for policy B against the numerical comparator with the lowest mean regret in each family (Table 23).

Family	Reported paired p	Bonferroni p (10 comparisons)
Sphere	2×10^{-9}	2×10^{-8}
DiffPowers	0.052	0.520
SchaffersF7	—	—
Katsuura	n.s.	—
Weierstrass	n.s.	—
Lunacek	0.029	0.290
SharpRidge	3×10^{-5}	3×10^{-4}
Rosenbrock-rot	6×10^{-4}	0.006
AttrSector	—	—
Discus	2×10^{-5}	2×10^{-4}

Table 25: Multiplicity sensitivity for the reported numeric paired tests. The Bonferroni column applies a factor of ten to match the ten-family comparison.

Five reported wins have nominal $p < .05$: Sphere, Lunacek, Sharp Ridge, rotated Rosenbrock, and Discus. Four survive the ten-test correction; Lunacek does not. Different Powers has $p = .052$ and does not meet .05 even before correction. Katsuura and Weierstrass are reported as nonsignificant. The similar Schaffers means (8.87 versus 8.83) do not establish equivalence. These interpretations rely on the reported tests, subject to the data and methodological limitations in Appendix E.

M BUDGET AND DIMENSION ENDPOINTS

Task	Budget	Base	A	B	v11	v14	GP-BO	evo	HEBO
Quadratic	20	69.5	36	21.4	30.5	24.2	30.4	18.5	41.5
Quadratic	50	31	14.6	11.7	6.6	9.8	14.9	5.5	4.8
Quadratic	100	9.6	14.7	9.1	3.6	3.1	8.4	3.8	0.79
Rastrigin	20	77.4	65.3	57	49.2	59.9	83.9	83.8	98.1
Rastrigin	50	46.1	36.7	26.6	3.4	18.2	67.6	59.5	63.6
Rastrigin	100	52.4	25.7	26.7	6.8	7.7	55.2	53.2	53.4

Table 26: Budget sweep on the two core tasks. Entries are mean final regrets over $N = 30$ instances per setting. These are evaluation endpoints at different configured budgets, not checkpoints on a common optimization trajectory.

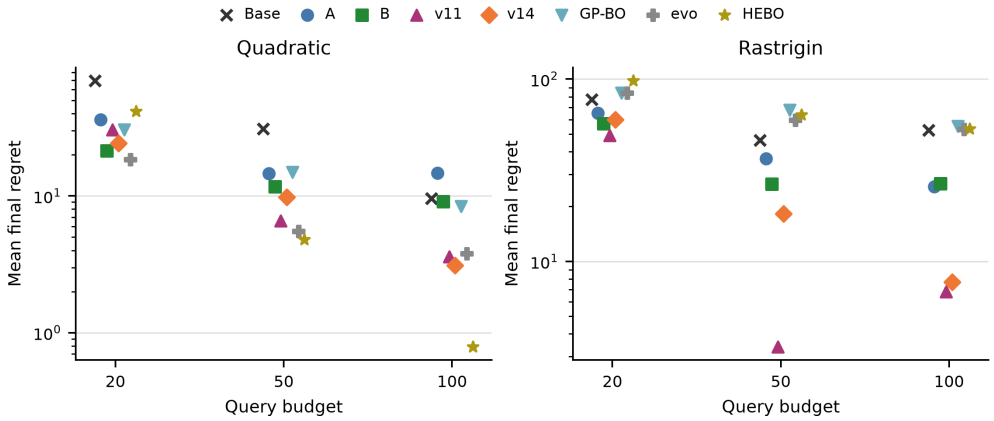

Figure 4: **Budget changes the relative ordering of methods.** Separately configured endpoints from Table 26; log regret axis, with horizontal offsets for legibility. These are not shared-prefix trajectories.

For any fixed sequence of evaluated points, best-so-far regret cannot increase when its prefix is extended. The increase of v11's Rastrigin mean from 3.4 at 50 queries to 6.8 at 100, and Base's increase from 46.1 to 52.4, show that the reported summaries cannot be used as a shared-prefix convergence curve. They establish comparisons within each budget setting; without paired trajectories across budgets, they do not identify why those increases occur.

Task ($d = 20$)	Base	A	B	v11	v14	evo	GP-BO
Quadratic	819.7	725	1,110.3	664.4	633	488.3	937.2
Rastrigin	317.2	295	360.4	272.8	283.1	299.3	351.4

Table 27: $d = 20$ stress test with $N = 30$ instances. Lower mean regret is better.

At $d = 20$, B is 35.5% above Base on quadratics and 13.6% above Base on Rastrigin. Meanwhile v11 reduces the respective means by 18.9% and 14.0%, and v14 by 22.8% and 10.8%. These are objective-specific relative changes, not a dimensional scaling law. For the separate $B = 100$ Rastrigin block, v11's 6.8 is compared to GP-BO's 55.2, evo's 53.2, and HEBO's 53.4; each reported paired test has $p < 10^{-8}$. All three retain bounds below 3×10^{-8} under a three-test Bonferroni correction. No significance claim is inferred for the quadratic or dimensional means without a supplied test.

N MECHANISM HETEROGENEITY ACROSS PROMOTED ARTIFACTS

The main text uses Harness A as the canonical mechanism case because its champion program is bare code. This criterion is structural: it removes the confound that explanatory comments can carry the same practice-derived information as the separate record. The full cross-champion results are reported here so that the representative case does not hide heterogeneity.

	Quadratic			Rastrigin		
Champion	Full	Code	Δ	Full	Code	Δ
A	24.4	33.4	+9.0	30.6	47.5	+16.9
B	22.9	22.8	-0.1	60.8	77.0	+16.2
v11	23.8	25.1	+1.3	66.2	45.0	-21.2
v14	23.1	25.1	+2.0	63.4	56.3	-7.1

Table 28: Complete Full–Code cross-champion ablation from the reported aggregate results. Full supplies code+record; Code omits the separate record; $\Delta = \mathrm{Code} - \mathrm{Full}$. Positive values favor Full.

Artifact	Quadratic contrast	Full	Ablated	Nominal p
A	Code-only (remove record)	24.4	33.4	.017
B	Record-only (remove program)	22.9	33.3	1.7×10^{-5}
v11	Stripped (remove record+comments)	23.8	32.0	.0001

Table 29: Carrier-focused quadratic ablations. Each condition uses two independent distillation generations and $N = 30$ paired instances on the Pro executor. The reported p-value compares Full with the listed ablation. For context, B scores 22.8 under Code-only and 23.0 under Stripped; v11 scores 25.1 under Code-only.

The quadratic additions provide complementary evidence about where practice-derived information can enter distillation. For A, removing the separate record degrades performance. For B, Code-only and Stripped remain essentially at Full (22.8/23.0 versus 22.9), whereas Record-only rises to 33.3; thus the record alone does not reproduce the information available in the program representation. For v11, Code-only remains close to Full (25.1 versus 23.8), while Stripped rises to 32.0. Because Stripped removes both the separate record and explanatory comments, and record removal alone is small in this block, this pattern is consistent with explanatory program text contributing to the distilled policy. These contrasts are artifact-specific mechanism evidence, not a claim that one carrier is universally necessary.

For A, the Full–Code contrast is directionally consistent on both tasks: 24.4 versus 33.4 on quadratics and 30.6 versus 47.5 on Rastrigin. Because A has no explanatory comments, this is the cleanest available test of the separate record channel. B gives a compatible off-family result on Rastrigin (60.8 versus 77.0, reported $p = .003$). A completed stripped-code variant for B scores 70.9 on Rastrigin ($p = .068$ versus Full), again moving away from Full when both the record and comments are removed. With only two distillation generations, these comparisons remain sensitive to generation noise.

The other champions demonstrate why we do not generalize the A result to every artifact. v11 reverses on Rastrigin (Full 66.2 versus Code 45.0), and v14 also does not show a consistent Full advantage. These are genuine counterexamples, not excluded failures. They support a carrier view: practice-derived information may reach the distiller through the separate record, through executable program structure, or through explanatory text embedded in the program.

The unadjusted paired tests for A are .017 and .026. In the original eight-comparison family their Holm-adjusted values are .102 and .130, so the main text treats the A result as a mechanism exhibit rather than a multiplicity-adjusted population claim. The comparisons that remain below .05 after that correction are B on Rastrigin, favoring Full, and v11 on Rastrigin, favoring Code. Multiplicity therefore does not erase the artifact-level heterogeneity.

The pooled means are 23.6 versus 26.6 on quadratics ($p = .009$) and 55.2 versus 56.5 on Rastrigin ($p = .98$). These pooled task-level values do not represent independent distillations: there are only

	Quadratic		Rastrigin	
Champion	Reported p	Holm p	Reported p	Holm p
A	.017	0.102	.026	0.130
B	.37	0.450	.003	0.021
v11	.15	0.450	<.001	<.008
v14	.22	0.450	.08	0.320

Table 30: Exploratory multiplicity sensitivity for the original Full–Code comparisons. Direction must be read from Table 28; v11's Rastrigin effect favors Code.

two generations per condition and champion. B's two code-only Rastrigin generations score 55.7 and 98.4, a 42.7-point spread, further showing that generation-level variability can be large.

What the ablation identifies. The ablation does not test whether executable practice itself is necessary: every Full and Code condition inherits a program produced by executable practice. Instead it asks which representation carries practice-derived information into distillation. The clean A case shows that the separate record can be load-bearing when the program is bare; the cross-champion counterexamples show that no single carrier is universally necessary. This is consistent with the paper's broader representation claim without requiring the four promoted artifacts to behave identically.

O PRODUCTION RECOMMENDATION TUNING CASE STUDY

We report an anonymized external-validity case study from a live commercial video recommendation platform serving hundreds of millions of daily active users (DAU), evaluating low-budget optimization ($B = 20$) on real industrial live A/B testing data (Table 31). The offline benchmark is calibrated from multi-month production logs of completed live A/B testing campaigns (with each experiment running for multiple days on live production traffic slices covering millions of DAU), tuning a seven-dimensional continuous candidate-blending configuration in $[-5.0, 5.0]^7$ initialized at the deployed production warm start x_0. Across these empirical live A/B testing benchmarks, we evaluate 30 original seeds, 30 fresh confirmation seeds, and an 81-seed pooled block. Because the proprietary engagement objective's normalization is not part of the public manuscript package, we do not compare raw regret magnitudes across benchmarks. "Code-to-Harness" denotes the production configuration of our approach used in this study.

Method	Original	Fresh	Pooled	Ours vs. comparator (wins; p)
Code-to-Harness	**0.152**	**0.145**	**0.143**	—
Base	0.236	—	—	25/30; .004
GP-BO	0.170	0.156	0.159	50/81; .22
Production GP-bandit	0.183	0.164	0.157	53/81; .22
CMA-ES	0.188	0.172	0.190	54/81; $< 10^{-4}$
Random	0.178	0.225	0.200	62/81; $< 10^{-4}$

Table 31: Production recommendation tuning case study. Entries are mean reported final regret at $B = 20$ on 7D empirical response surfaces calibrated from industrial live A/B testing logs; lower is better. Original (1–30) and fresh (31–60) sets contain 30 distinct seeds each. Pooled covers the 81 paired seeds shared by Code-to-Harness and all non-LLM optimizers. The last column counts paired seeds on which Code-to-Harness is lower. For the non-LLM family, the supplied record reports Holm-adjusted p-values; the Base comparison is reported separately. Dashes denote configurations not run.

On the 7D offline production surrogate, Code-to-Harness achieves the lowest mean regret on the original, fresh, and pooled blocks ($0.152/0.145/0.143$), outperforming Base on the original 30 seeds (0.236; $25/30$ wins, $p = .004$). Over the 81-seed pooled block, Code-to-Harness significantly improves over Random (0.200; paired $t = -5.31$, Holm-adjusted $p < 10^{-4}$; $62/81$ wins) and CMA-ES (0.190; $t = -4.26$, $p < 10^{-4}$; $54/81$ wins), while matching GP-BO (0.159; $50/81$) and the platform's production GP-bandit service (0.157; $53/81$; both Holm-adjusted $p = .22$). In a separate live online A/B traffic tuning campaign for a production ranking parameter (each evaluation round runs for 1–2 weeks on live user traffic), Code-to-Harness reached the launched configuration in two rounds, compared with four rounds for a prior in-house CMA-ES cycle to reach its first positive candidate. In the confirming live A/B test (7 days, roughly 7.5 million DAU per arm), the launched configuration increased module-level valued engagement time per DAU by $+0.39\%$ (95% CI $[+0.30, +0.49]$) and sitewide valued engagement time per DAU by $+0.16\%$ (95% CI $[+0.10, +0.22]$); it has since been rolled out to all production traffic.