RETHINKING AGENT EVOLUTION: DO EVOLVED GUI HARNESSES ACTUALLY OUTPERFORM DIRECT INFERENCE?

Anonymous authors
Paper under double-blind review

ABSTRACT

Program evolution is increasingly used to improve language-model agents, with progress typically measured against the initial program. We study whether seed-relative gains translate into an advantage over well-specified direct interfaces to the same frozen model. We introduce EvoHarness, a framework for evolving GUI grounding harnesses and auditing the resulting gains with frozen interface controls, matched execution backends, and repeated measurement. EvoHarness searches over executable preprocessing, prompting, parsing, and routing code, retaining execution traces for post hoc analysis. In a pilot search, evolution finds a harness that reaches 89.15% accuracy on all 1,272 ScreenSpot examples. Under matched backends, however, the evolved champion does not consistently outperform direct inference controls on the same frozen model: on Qwen, it improves from 20.33% to 54.33% over the seed but remains below a single-call direct-point baseline at 70.67% (paired Holm $p = 1.26 \times 10^{-5}$); on Kimi, it is not significantly better than either direct control. Repeated evaluation of identical code also yields apparent gains: a gain-only rule accepts 5/20 disjoint same-code pairs, while a paired gate accepts 0/20. These results suggest that seed-relative gains alone are insufficient evidence for agent improvement. We recommend frozen direct-interface controls, matched execution resources, and uncertainty reported at both the program and search levels.

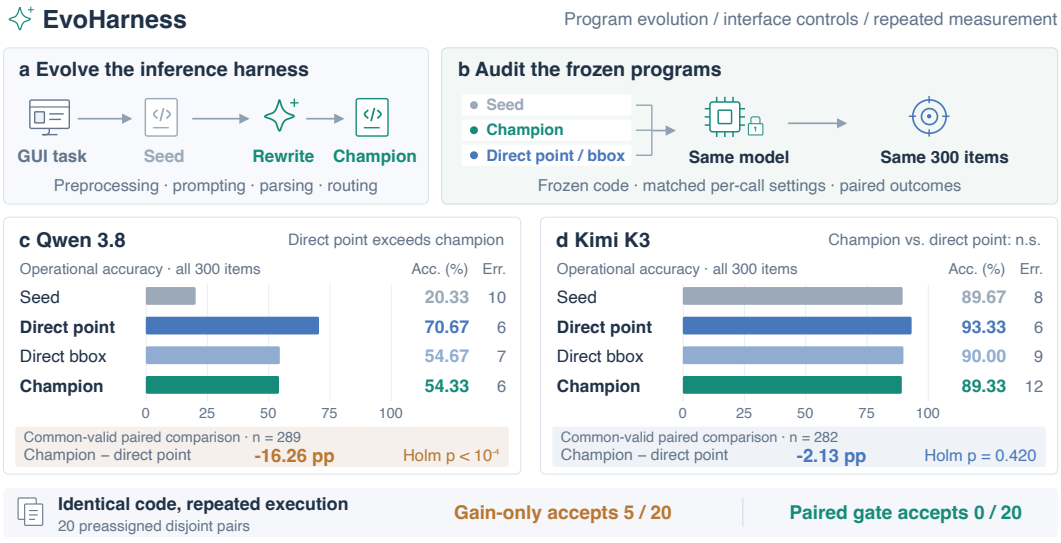


Figure 1: EvoHarness evolves GUI harnesses and audits frozen programs against direct interfaces. Bars show accuracy on 300 ScreenSpot items per backend, with execution errors counted as misses (Err.). Paired differences and Holm tests use common error-free items (Qwen: 289; Kimi: 282), so they need not equal bar differences. The same-code audit uses 20 disjoint execution pairs.

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

1 INTRODUCTION

GUI agents map a screenshot and instruction to an action, often by predicting a point inside a target element (Cheng et al., 2024; Wu et al., 2024; Gou et al., 2024; Xu et al., 2024). Their performance depends not only on model weights, but also on the surrounding inference interface: image resizing, prompt format, coordinate conventions, parsing, retry logic, and model routing can all affect whether a useful prediction becomes a correct action. Changes to the harness around a frozen model are therefore part of the evaluated system.

Program evolution can automate these choices by searching over executable agent code (Zhang et al., 2025a; Hu et al., 2024), workflows (Zhang et al., 2024), or more general computational structures (Romera-Paredes et al., 2024; Novikov et al., 2025). Prompt optimization explores related changes at the instruction level (Fernando et al., 2023; Guo et al., 2024; Yang et al., 2023). In GUI grounding, inference-time visual search further shows that how a model is queried can affect its predictions (Li et al., 2025a; Lian et al., 2025). A gain over an initial harness thus measures improvement of a particular starting system; it does not by itself identify the value of the evolved program relative to a direct interface.

When evaluation relies on seed-relative gains and repeatedly reused selection data, it mixes two questions:

1. Did evolution improve the program beyond a well-specified direct interface to the same model?
2. Did the search reliably find a better program, or did repeated reuse of the dev set select a favorable measurement?

These questions interact because interface choices can affect both mean accuracy and execution variability.

We study them with *EvoHarness*, which evolves runnable Python harnesses for frozen vision models and retains proposed programs, per-example traces, and selection decisions. The evaluation combines frozen direct-interface controls, matched execution backends, and repeated measurements of identical code. A motivating pilot found a strong transferred champion, but later controlled comparisons changed its interpretation.

On 300 ScreenSpot examples with Qwen, the champion improves over the seed by 34.0 percentage points but trails a single-call direct-point baseline by 16.33 points (Figure 1). On Kimi, it has no statistically significant advantage over either direct control. These comparisons evaluate frozen transferred programs, rather than searches rerun on each endpoint. A separate same-code audit examines whether repeated execution alone can produce gains large enough to trigger selection.

Our contributions are:

- We present a controlled case study of GUI harness evolution showing that large seed-to-champion gains can coexist with worse performance than simple direct-inference baselines under matched backends.
- We quantify repeated-measurement instability for identical code and show that gain-only acceptance can admit spurious improvements.
- We propose an evaluation protocol for evolved agents based on frozen interface controls, matched execution resources, and uncertainty reported at both the program and search levels.

2 RELATED WORK

Program and agent evolution. Search over prompts, workflows, and executable agent code offers several ways to improve language-model systems (Fernando et al., 2023; Guo et al., 2024; Yang et al., 2023; Zhang et al., 2024; Hu et al., 2024; Zhang et al., 2025a). Evaluation-guided program discovery extends this approach to computational tasks (Romera-Paredes et al., 2024; Novikov et al., 2025). Our setting is closest to code-level agent evolution, but our goal is to interpret gains when search starts from a potentially weak harness and repeatedly reuses selection data. Work attributing prompt-harness gains to individual slots (Nguyen et al., 2026) is complementary: we compare frozen executable programs with explicit direct coordinate interfaces.

GUI grounding and inference-time policies. GUI-specific training improves screenshot grounding and interaction (Hong et al., 2024; Cheng et al., 2024; Wu et al., 2024; Gou et al., 2024; Xu et al., 2024; Qin et al., 2025; Wu et al., 2025). Inference-time focus, decomposition, and cropping

Table 1: Mutation contexts. All protocols start from the same seed; their accepted programs may subsequently diverge.

Protocol	ID	Failure diagnosis	Improvement directive
Guided	A	included	included
Blind	B	withheld	included
Random	R	withheld	withheld

A/B/R identify artifact runs. The pilot's first-generation operators differed across arms; a separate matched-operator control is described in appendix D.

alter how visual evidence reaches the model (Li et al., 2025a; Lian et al., 2025; Yang et al., 2025; Fu et al., 2026), while spatial diagnostics and high-resolution benchmarks expose additional challenges (Jahin and Parvez, 2026; Li et al., 2025b). EvoHarness keeps weights frozen and searches over the surrounding program. We examine how output contracts and execution backends affect the interpretation of its gains, rather than propose a new grounding model or visual search policy.

Evaluation reliability and benchmark auditing. Repeated trials, implementation controls, and uncertainty estimates are central to reliable empirical comparisons (Henderson et al., 2018; Engstrom et al., 2020; Agarwal et al., 2021). Benchmark audits also examine the measurement procedure and its assumptions (Kiela et al., 2021; Raji et al., 2020). Recent agent studies distinguish scaffold effects from scoring validity (Zhang et al., 2026) and analyze paired leaderboard resolution (Liu et al., 2026). We study these concerns in GUI program evolution through fresh executions of identical code and frozen interface controls. Exact paired testing (McNemar, 1947; Dietterich, 1998) provides a candidate filter; our audit separately examines its observed behavior without claiming error control over an adaptive search.

3 EvoHarness: Program Search with Auditable Selection

3.1 Search object and mutation context

A harness is a Python module with an interface `solve(sample, models)` returning a point prediction and an execution trace. The sample passed to the harness contains only an image and an instruction. The program may preprocess images, construct prompts, parse model outputs, crop or zoom, and choose between the exposed frozen vision models. The seed is a single-call program with free-form coordinate parsing. A full-source rewrite creates each candidate, rather than editing a fixed declarative workflow.

Candidates are full Python rewrites subject to a restricted execution environment. Image processing and ordinary computation are allowed, while file access, subprocesses, unrestricted network calls, and reflective access are blocked by a code filter. Ground-truth boxes remain in the external evaluator; the access rules and their limitations are in appendix A.

The mutation prompt contains parent source, an interface contract, and the parent's aggregate selection accuracy and cost. In the guided protocol it also includes failing instructions, predictions, ground-truth boxes, category-level outcomes, and a vision-model diagnosis of four failures. The blind protocol replaces this block with a placeholder; the random protocol additionally removes the improvement directive. "Blind" thus means no failure-level feedback, not absence of task knowledge, parent code, or aggregate scores. Repair attempts receive exception text only.

3.2 Execution, cost, and candidate selection

One generation proposes two candidates from the current parent. A four-example smoke evaluation catches code failures and excessive cost before full dev evaluation, with at most two exception-based repair passes. The smoke cost contract caps at three times the parent's per-example cost; full-evaluation cost is logged separately. Endpoint failures are distinguished from deterministic code failures, and a generation blocked entirely by infrastructure is retried rather than recorded as search failure.

For a harness h and sample s, cost sums model-call token usage with a gateway-specific price τ_j in cost units (CU) per thousand tokens:

$$c(h,s) = \sum_{j=1}^{m(h,s)} \frac{t_j}{1000}\tau_j, \qquad \mathrm{cps}(h,S) = \frac{1}{|S|}\sum_{s\in S} c(h,s). \tag{1}$$

The historical search uses accuracy as fitness ($\lambda = 0$ in $\text{fit}_\lambda = \text{acc} - \lambda\text{cps}$). It therefore does not directly optimize a cost–accuracy objective. A cheaper accepted program is an observed outcome of mutation and selection; the smoke contract limits expenditure but does not establish a global Pareto optimum.

For each evaluation, accuracy is computed on successfully measured examples and the error rate is reported separately. A run is invalid when its error rate exceeds 20% or its total recorded cost is zero. For paired inference, we use only examples measured successfully for both parent and candidate. Let b count parent-hit/child-miss outcomes, and c parent-miss/child-hit outcomes. The historical acceptance rule, called ValidGate, is

$$\Delta\text{acc} > 0.03 \quad \wedge \quad p_{\text{exact}}(b, c) < 0.05 \quad \wedge \quad c > b, \tag{2}$$

where

$$p_{\text{exact}}(b, c) = \min\left(1, 2\sum_{i=0}^{\min(b,c)} \binom{b+c}{i} 2^{-(b+c)}\right). \tag{3}$$

This is the two-sided exact McNemar test (McNemar, 1947; Dietterich, 1998). We first choose the valid candidate with highest fitness and then apply the paired gate; at most one candidate replaces the parent. The gain threshold uses each run's valid-sample accuracy, while the test uses common valid items. Repeated candidate selection and reuse of dev data mean that the nominal p threshold does not provide search-wide error control. The full selection procedure and denominator details appear in appendix A.

3.3 Acceptance resolution and statistical power

We distinguish the smallest observed gain that can satisfy the gate from the probability of detecting a true gain. With small evaluation sets, an observed gain can cross the acceptance threshold even when power to detect such gains is low. For example, under an illustrative independent-item model with discordance probability 0.30, a true 10-point gain has power 0.133 at $n = 40$ and 0.703 at $n = 200$. Appendix B gives the assumptions, derivation, and power curves (Figure 5).

4 Experimental Setup

Synthetic selection and held-out data. SynthGUI procedurally renders seven element classes. The original splits contain 40 examples each: dev and test use a 390×720 light mobile theme, while the dark-theme splits use 720×460 web layouts. Median target area is 0.93% of the dev canvas and 0.47% of the held-out dark canvas. Selection uses dev only; test and test_dark are held out from mutation and selection. The small synthetic setting exposes program and evaluation behavior rather than representing the full diversity of GUI tasks.

Real screenshots. ScreenSpot v1 contains 1,272 grounding examples across mobile, desktop, and web interfaces, with text and icon targets (Cheng et al., 2024). Evaluation uses the point-in-box criterion. No ScreenSpot image, instruction, or label enters the original search. The historical champion was evaluated on all 1,272 examples; the seed was first compared on a fixed 60-example subset and, after the campaign, was measured on the full benchmark as well. These are distinct evaluations of the same grounding objective.

Models and budgets. Historical harnesses access a strong model through `qwen_vl_max` and a cheaper model through `ling_flash_vl`; their served model identifiers are `Qwen3.8-2.4T-A95B-SPOT` and `ling_3_flash_vl`. Token accounting uses $\tau = 4$ and $\tau = 1$ CU per thousand tokens, respectively. These are the study's gateway accounting weights, not universal monetary prices. Mutation operators run at temperature zero, which does not make the remote service deterministic. The pilot lasts three generations; the subsequent replication plan uses five generations and eight complete runs per arm under 4500-CU and eight-hour guards, with completion status reported below. Run identifiers index repetitions, not seed-controllable LLM sampling. Additional implementation settings are in Table 7.

Comparisons and reporting units. We separate three sources of evidence: one original search trajectory, repeated executions of fixed code, and repeated search runs. The first establishes that a program was discovered, the second describes measurement stability, and the third assesses the

search procedure. Accuracy repeats are not interchangeable with independent searches, and historical paired tests describe their specific evaluations rather than corrected guarantees over all adaptive search decisions.

5 RESULTS

We first compare the frozen champion with direct interfaces under matched backends, then examine its transfer and program changes. Repeated executions separate program quality from measurement variability; repeated searches provide a further, currently incomplete, level of evidence.

5.1 MATCHED-BACKEND COMPARISONS

We freeze the seed, the evolved champion, and two single-call controls that request normalized point or bbox JSON. The controls specify both prompt and output contract. Within each backend, both model aliases map to one served model (`qwen3.8-2.4t-a95b-spot` or `kimi-k3-spot`), preserving the champion's code, preprocessing, and routing. All four programs use the same 300 frozen ScreenSpot IDs, interleaved execution, temperature zero, and an 8192-token cap per call. We evaluate each program once per item. These are transferred programs, not searches rerun on the new endpoints; their results are analyzed separately from the original gateway.

The champion's 34.0-point gain over the seed on Qwen coexists with a 16.33-point deficit to direct point (Table 2). On common error-free items, the paired deficit is 16.26 points (Holm $p = 1.26 \times 10^{-5}$). Against direct bbox, the paired difference is small and nonsignificant. The table separates all-item operational accuracy from common-valid paired statistics so that execution failures remain visible.

Table 2: Frozen programs under matched backends. Top: operational accuracy on all 300 items, with execution errors counted as misses. Bottom: champion-minus-control differences on common error-free items.

	Qwen 3.8		Kimi K3	
Harness	Acc. (%)	Errors	Acc. (%)	Errors
Original seed	20.33	10	89.67	8
Direct point	**70.67**	6	**93.33**	6
Direct bbox	54.67	7	90.00	9
Evolved champion	54.33	6	89.33	12

Backend	Champion vs.	Paired n	Δ (pp)	Holm p
Qwen 3.8	Direct point	289	-16.26	1.26×10^{-5}
Qwen 3.8	Direct bbox	289	-0.35	1.000
Kimi K3	Direct point	282	-2.13	0.420
Kimi K3	Direct bbox	279	-0.36	1.000

Bold marks the highest observed operational accuracy. Exact McNemar tests use Holm adjustment across two control comparisons per backend. Pair counts are in appendix I.

On Kimi, direct point reaches 93.33% operational accuracy and the champion 89.33%. Their common-valid paired difference is -2.13 points (Holm $p = 0.420$); the champion also has no significant advantage over direct bbox. Nonsignificance does not establish equivalence or the absence of a useful effect. Across the two tested backends, the transferred champion has no demonstrated advantage over direct point.

Matched per-call settings do not imply equal realized resource use (Table 3). The champion invokes additional calls on some items, while its mean token use is below direct point on Qwen and above it on Kimi. The comparison therefore shows no uniform resource advantage for the champion.

5.2 TRANSFER AND PROGRAM CHANGES

The direct-control comparison gives context to the strong seed-relative gains on the original two-model stack (Table 4). The pilot champion reaches 95.0% on held-out SynthGUI-test, 100.0% on the unseen dark theme, and 89.15% on all 1,272 ScreenSpot examples, versus the seed's 52.5%, 36.25%, and 21.14%, respectively. The full-benchmark paired comparison strongly favors the champion ($p = 2 \times 10^{-230}$). These results establish transfer beyond the synthetic selection set.

Table 3: Inference resources under matched backends, excluding search. Calls average logged attempts over all 300 items; tokens use the accounting subset defined below.

	Qwen 3.8		Kimi K3	
Harness	Calls/ex.	Tokens/ex.*	Calls/ex.	Tokens/ex.*
Original seed	1.00	3323	1.00	3994
Direct point	1.00	4719	1.00	4070
Direct bbox	1.00	4702	1.00	4268
Evolved champion	1.08†	4158	1.06	4671

*Mean over error-free evaluations with complete usage accounting, including grounding misses. †Lower bound: two failed Qwen champion attempts lack call logs.

Table 4: Transfer on the original two-model execution stack. Accuracy (%) is reported per evaluation or as mean ± sample SD over R executions. Selection and held-out evaluations are separated.

Evaluation	R (seed/champ.)	Seed	Champion	Champion CU/ex.
Selection set				
SynthGUI-dev	9/3	47.50 ± 5.8	100.00 ± 0.0	0.618
Held-out synthetic				
SynthGUI-test	2/2	52.50 ± 0.0	95.00 ± 0.0	0.707
SynthGUI-test_dark	2/2	36.25 ± 5.3	100.00 ± 0.0	0.667
Real screenshots				
ScreenSpot (paired $n = 60$)	1/1	21.67	96.67	3.113
ScreenSpot (full $n = 1272$)	1/1	21.14	89.15	3.097

Four full-benchmark seed errors are excluded from its accuracy denominator and counted as misses in the paired test. CU/ex. reports champion inference cost only. Pair identities and tests are in appendix C.

Figure 2 shows two recovered targets, the sole regression, and the common failure on the 60-example paired subset. The regression misses the target's left boundary by 4.08 source-image pixels, illustrating how a small spatial error changes the binary grounding outcome. Case identities and selection are documented in appendix C.

The accepted trajectory changes both the output contract and model routing (Table 5). A structured bbox and crop/verification cascade first increase accuracy at higher cost. Cheap-first confidence routing then produces the retained champion: all 40 selection examples exceed its 0.85 first-stage threshold and use one cheap call. The 11-fold inference-cost reduction is relative to the seed, excluding search.

Table 5: Program changes along the blind pilot ($n = 40$). Rows report the seed and each evaluated child; paired counts compare child with parent. Bold identifies the retained champion.

Gen.	Program change	Acc. (%)	CU/ex.	Fix/break	p	Decision
0	Free-form coordinate seed	45.0	6.940	–	–	Initial parent
1	Bbox contract; crop/verify	75.0	18.450	15/3	0.0075	Accept
2	**Cheap-first; confidence routing**	**100.0**	**0.618**	10/0	0.0020	Accept
3	Unaccepted challenger	100.0	0.800	0/0	1.0000	Keep parent

CU/ex. excludes search expenditure. This is one trajectory, not an ablation or an average over searches.

The coordinate contract is a likely source of the seed's weakness. Its parser scales unit-interval coordinates but otherwise reads them as pixels; four inspected failures contain outputs on a $[0, 1000)$ scale. The champion explicitly requests normalized bbox JSON and also changes preprocessing and the first-stage model. Figure 3 separates these changes and shows that additional model calls are conditional. This is a structural comparison, not an ablation assigning the gain to a single edit.

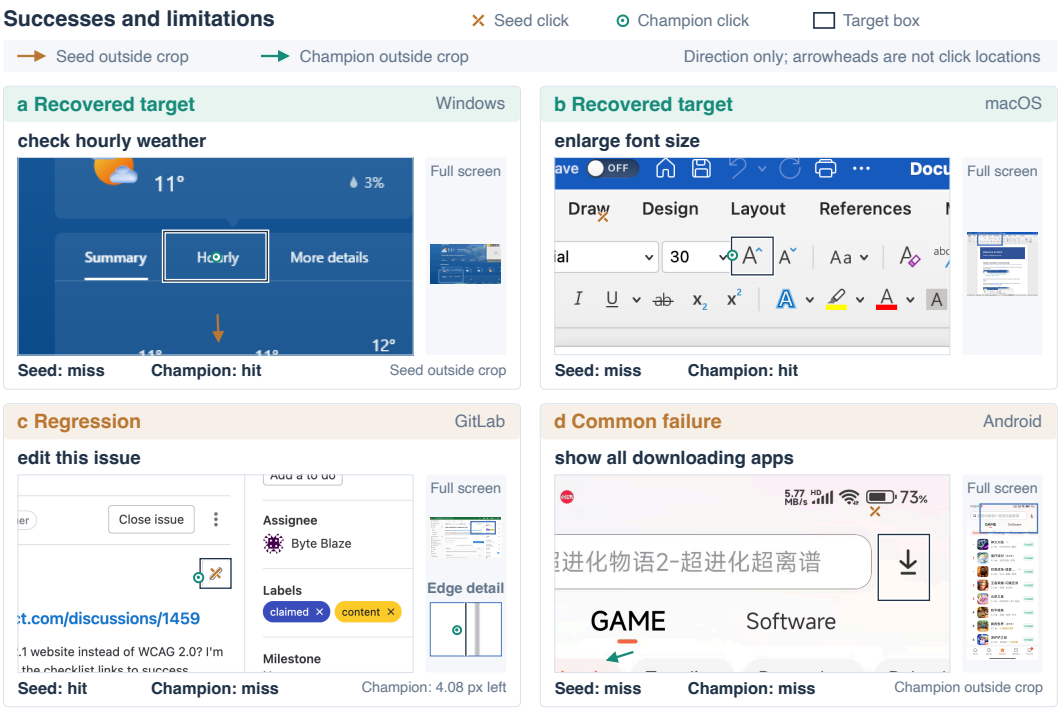


Figure 2: Paired ScreenSpot cases: two recoveries, one regression, and a shared failure. Boxes mark targets; arrows give directions to off-crop predictions, not click locations. Blue outlines locate crops in full-screen views. The regression detail magnifies the champion's boundary miss.

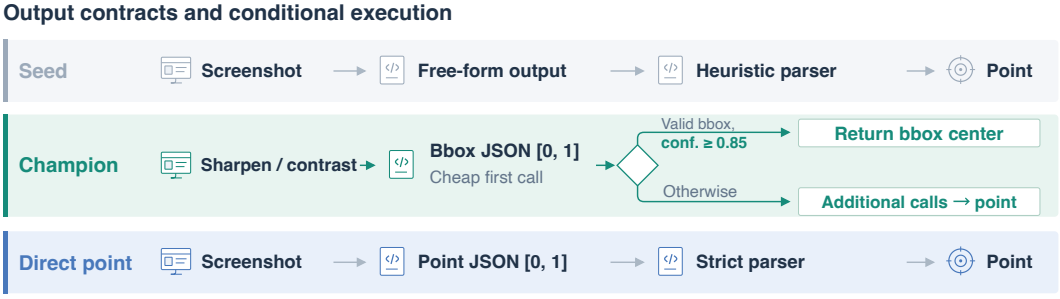


Figure 3: Output contracts and execution paths. The champion returns directly when the first bbox is sufficiently confident; otherwise it uses additional model calls. In the matched-backend study, both aliases map to the same model.

5.3 Same-code measurement variability

Identical code can produce apparent improvements under repeated evaluation. In a post-freeze audit, 40 seed executions on the same 40 dev items form 20 preassigned disjoint pairs. Gain-only selection accepts 5/20; the paired gate accepts 0/20 (Table 6 and Figure 4a). Because the code is unchanged, these gains cannot represent program improvement.

All fresh pairs pass the error-rate validity check. Gains use each run's error-free denominator; paired tests use common valid items. Each execution appears in one pair, with orientation fixed in advance. Shared screenshots and service drift remain, so zero acceptances in 20 pairs do not establish a 5% false-positive bound or calibration during adaptive search. The earlier audit reuses executions and helped motivate the gate; its acceptances provide descriptive evidence rather than independent calibration (appendix J).

On 200 synthetic items, five executions per program yield $42.3 \pm 2.0\%$ for the seed and $61.7 \pm 3.1\%$ for the champion (mean $\pm$ sample SD; Figure 4b). Their ranges are 41.0–45.5% and 58.5–65.0%. The champion-over-seed gain persists, but both programs vary. The original champion's repeated

Table 6: Same-code acceptances and audit design. Counts describe decisions under each protocol, not independent false-positive rates.

	Historical audit	Post-freeze audit
Comparison pairs	38	20
Executions reused across pairs	Yes	No
Pair direction fixed in advance	No	Yes
Gain-only (Δacc > 0.03)	17/38	5/20
Paired gate (Eq. (2))	2/38	0/20

Both audits share items and service conditions across comparisons. Protocols and backends differ; their counts are not pooled.

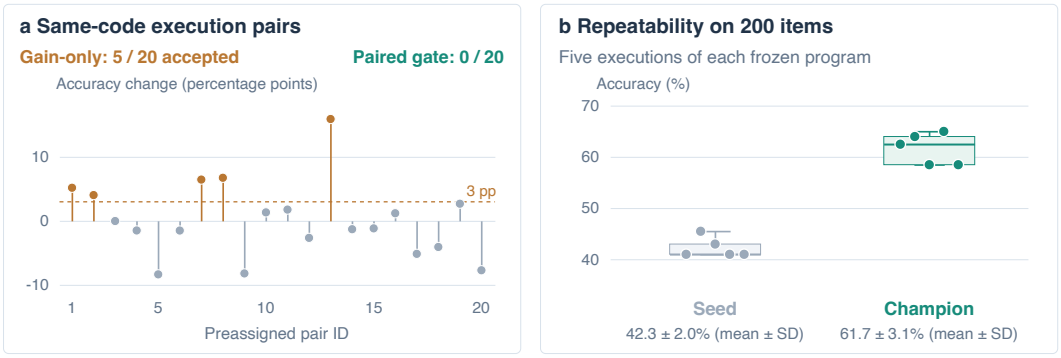


Figure 4: Repeated execution on Qwen. (a) Same-code gains for 20 disjoint pairs; orange marks gain-only acceptances above 3 points. The paired gate accepts none. (b) All five dev200 repeats per program, with errors counted as misses. Boxes show the median and interquartile range; whiskers show the observed range.

synthetic accuracies were identical on small, partly saturated sets (Table 4); the larger-set results show why those observations do not imply deterministic execution.

5.4 Search replication and scope

Program-level results do not establish how reliably a search finds improvements. The three-generation pilot accepts two blind mutations and no guided mutation, but later runs do not support a strong claim that diagnosis harms search. In the 18 September 2026 snapshot, all six completed guided runs and six of eight completed blind runs have an acceptance (two-sided Fisher $p = 0.47$). Completion is unequal and the planned replication remains unfinished. We treat the pilot contrast as anecdotal; operator controls and full run accounting are in appendix D. Service failures further affect which evaluations and searches complete; the observed incidents and safeguards are documented in appendix E.

6 Discussion and Limitations

The central distinction is between a successful discovered artifact and evidence for a search procedure. The historical champion transfers well on its evaluated stack, yet the matched-backend comparison shows the same evolved source underperforming a simple explicit-point interface; the 34.0-point gain over the seed alone would conceal this deficit. Direct contract controls are an important baseline for program search, especially when the initial harness may be weak or underspecified. Contracts must be measured, not merely written into prompts: both direct controls produce contract-violating responses, counted as misses without rescoring (appendix H); the point baseline's advantage holds with its recorded contract failures, while the bbox result cannot isolate a pure box-versus-point effect.

The feedback and same-code audits concern different levels of evidence: accepted guided mutations in later runs constrain the pilot's negative result, while shared-run comparisons constrain claims of calibrated false acceptance. Both favor reporting fixed-program quality, execution variability, and independent-search outcomes separately.

Several limitations remain. The historical search uses one small synthetic selection task, and real-data transfer evaluates the same grounding objective, not a second agent capability. Model identifiers are served endpoints, not immutable checkpoints. The pilot changes execution backends; the matched studies isolate that issue but use new served resources and one execution per program–item combination. Replication is unfinished, and aborted runs can make completed-run statistics selective. Finally, near-saturated aggregate accuracy has limited resolution for execution variance; larger independent sets and post-selection remeasurement are needed to assess stable improvements.

7 Conclusion

We show that, for GUI harness evolution, large gains over the seed do not necessarily imply better inference than simple direct-interface baselines on the same frozen model. In our case study, matched-backend comparisons change the interpretation suggested by the pilot trajectory alone, while repeated evaluation reveals substantial same-code variability. We therefore recommend reporting frozen direct controls, matched execution resources, and uncertainty at both the program and search levels when evaluating evolved agents.

Reproducibility statement

The local artifact package contains program lineages, per-example traces, acceptance decisions, frozen sample IDs, model identifiers, and figure-generation scripts. The supplementary protocol distinguishes historical measurements, the dated replication snapshot, and the matched-backend comparison. All new tables and plots are generated from these artifacts; credentials are excluded. An anonymous archive with the same files is required for submission. Implementation details and limitations of artifact portability are described in the appendix.

Ethics statement

SynthGUI is procedurally generated and ScreenSpot is a public screenshot benchmark. No new human-subject data were collected. Screenshots may contain interface text or user-generated content already present in the benchmark; data and redistribution must follow its license. Model calls consume remote inference resources. The code filter constrains the tested harness interface but is not a general-purpose security boundary for arbitrary generated programs.

AI use statement

Language models are used as mutation operators, frozen visual predictors, and failure diagnosers in the experiments. An AI coding assistant also contributed to hypothesis refinement, experimental design, implementation, statistical formulation and analysis, interpretation of results, and manuscript review and revision. It assisted with literature and citation checks and with the code for scientific figures and artifacts. SynthGUI images are procedurally rendered; no image-generation model supplies experimental screenshots. Numerical results are computed from stored executions or explicitly stated analytic models. Automated checks verify trace accounting and statistical calculations; they do not substitute for author review. Authors remain responsible for checking the code, citations, analysis, and claims.

References

Agarwal, R., Schwarzer, M., Castro, P.S., Courville, A., Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *NeurIPS* 34, 2021.

Chen, X., et al. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

Cheng, K., Sun, Q., Chu, Y., Xu, F., Li, Y., Zhang, J., Wu, Z. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. *arXiv preprint arXiv:2401.10935*, 2024.

Clopper, C.J., Pearson, E.S. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* 26(4), 404–413, 1934.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., Su, Y. Mind2Web: Towards a generalist agent for the web. *NeurIPS* 36, 28091–28114, 2023. arXiv:2306.06070.

Dietterich, T.G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation* 10, 1895–1923, 1998.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., Madry, A. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *ICLR* 2020. arXiv:2005.12729.

Fernando, C., et al. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Fu, Z., Wang, S., Zhang, W., et al. GUI-Lens: Coarse-to-fine cropping for GUI grounding with general-purpose VLMs. *arXiv preprint arXiv:2608.03270*, 2026.

Gou, B., Wang, R., Zheng, B., Xie, Y., Chang, C., Shu, Y., Sun, H., Su, Y. Navigating the digital world as humans do: Universal visual grounding for GUI agents. *arXiv preprint arXiv:2410.05243*, 2024.

Guo, Q., et al. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers (EvoPrompt). *ICLR* 2024. arXiv:2309.08532.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D. Deep reinforcement learning that matters. *AAAI* 2018. arXiv:1709.06560.

Hong, W., Wang, W., Lv, Q., Xu, J., et al. CogAgent: A visual language model for GUI agents. *CVPR* 2024 (Highlight). arXiv:2312.08914.

Hu, S., Lu, C., Clune, J. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Jahin, M.A., Parvez, M.R. GUI-Primitives: Diagnosing spatial reasoning failures in vision-language GUI grounding. *arXiv preprint arXiv:2608.21832*, 2026.

Kiela, D., et al. Dynabench: Rethinking benchmarking in NLP. *NAACL* 2021. arXiv:2104.14337.

Li, G., Xu, J., Zhao, Y., and Peng, Y. DyFo: A training-free dynamic focus visual search for enhancing LMMs in fine-grained visual understanding. *arXiv preprint arXiv:2504.14920*, 2025.

Li, K., Meng, Z., Lin, H., Luo, Z., Tian, Y., Ma, J., Huang, Z., Chua, T.-S. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*, 2025.

Lian, S., et al. UI-AGILE: Advancing GUI agents with effective reinforcement learning and precise inference-time grounding. *arXiv preprint arXiv:2507.22025*, 2025.

Liu, F., Liu, Y., Sun, R., Luo, L., Guo, S. Coding agents have converged: Why the SWE-bench leaderboard can no longer order its top entries, and what to measure instead. *arXiv preprint arXiv:2609.17394*, 2026.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., et al. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12(2), 1947.

Nguyen, M., Tan, W.C., Hassan, N.A., et al. Where does harness-optimization value live? Localized gains and the budget-splitting trap in self-evolving LLM agents. *arXiv preprint arXiv:2609.02889*, 2026.

Novikov, A., et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *Technical report, Google DeepMind*, 2025.

Qin, Y., Ye, Y., Fang, J., et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Raji, I.D., et al. Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. *FAccT* 2020. arXiv:2001.00973.

Renze, M., Guven, E. The effect of sampling temperature on problem solving in large language models. *arXiv preprint arXiv:2402.05201*, 2024.

Romera-Paredes, B., et al. Mathematical discoveries from program search with large language models. *Nature* 625, 2024.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., Yao, S. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Wang, G., et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, P., et al. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023.

Wang, W., Huang, Q., Xue, G., Liang, X., Li, J. LASER: Learning active perception via self-evolving preference optimization for GUI grounding. *arXiv preprint arXiv:2509.04243*, 2025.

Wilson, E.B. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association* 22, 209–212, 1927.

Wu, Z., et al. OS-Atlas: A foundation action model for generalist GUI agents. *arXiv preprint arXiv:2410.23218*, 2024.

Wu, Q., et al. GUI-Actor: Coordinate-free visual grounding for GUI agents. *arXiv preprint arXiv:2506.03143*, 2025.

Xu, Y., Wang, Z., Wang, J., Lu, D., Xie, T., Saha, A., Sahoo, D., Yu, T., Xiong, C. Aguvis: Unified pure vision agents for autonomous GUI interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Yang, C., et al. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2023.

Yang, Y., Li, D., Dai, Y., et al. GTA1: GUI test-time scaling agent. *arXiv preprint arXiv:2507.05791*, 2025.

Zhang, C., Yang, Z., Liu, J., Li, Y., Han, Y., Chen, X., Huang, Z., Fu, B., Yu, G. AppAgent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.

Zhang, J., et al. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

Zhang, J., Hu, S., Lu, C., Lange, R., Clune, J. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Zhang, X., Gao, Z., Zhang, B., et al. Adaptive Chain-of-Focus reasoning via dynamic visual search and zooming for efficient VLMs. *arXiv preprint arXiv:2505.15436*, 2025.

Zhang, Y., Motaali, S., Dinh, V.P., et al. The double measurement confound in agent benchmarks: De-scaffolding, ground-truth scoring, and reliability beyond the mean. *arXiv preprint arXiv:2609.09218*, 2026.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *NeurIPS* 2023. arXiv:2306.05685.

Zhou, S., Xu, F.F., Zhu, H., et al. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Execution details and pilot lineage

Candidate code passes an AST-based import and operation filter. Permitted libraries support image processing and ordinary computation; file access, subprocess execution, network access outside the model interface, and reflection are rejected. The filter caught a development-time attempt to import the benchmark generator. It enforces the tested access contract but is not a formal security proof for arbitrary Python code. Ground-truth boxes remain in the external evaluator.

The implementation first chooses the valid candidate with highest fitness, then tests that candidate; it does not test every candidate and choose among those passing. At most one candidate replaces the parent (Algorithm 1). The lineage is consequently a single path rather than a population.

Two qualifications matter. The historical gain compares each run's valid-sample accuracy, whereas b, c use their common valid subset; these estimands coincide only when the valid sample sets agree, where $\Delta\mathrm{acc} = (c - b)/n$ makes the direction clause redundant with positive gain—a defensive implementation check, not a separate statistical contribution. Moreover, choosing candidates and repeatedly reusing dev data is adaptive: a nominal per-comparison p threshold is not a guarantee of 5% false acceptance over the complete search.

Algorithm 1 One generation, matching the historical implementation.

Require: parent h, selection split S, mutation context u, $K = 2$
load or evaluate parent traces; reject infrastructure-invalid measurements
for $k = 1, \ldots, K$ **do**
    propose a full-source candidate h'_k using parent code and context u
    apply code filter, smoke evaluation, exception repair, and smoke cost check
    if candidate passes execution checks **then**
        evaluate h'_k on S; retain traces and error validity status
    end if
end for
$V \leftarrow$ candidates with valid full evaluation
if $V = \emptyset$ **then**
    return h (or retry if all failures are infrastructure-related)
end if
$h^* \leftarrow \arg\max_{h' \in V} \mathrm{fit}_\lambda(h', S)$
form paired counts for h^* and h on common successfully measured examples
if h^* passes Eq. (2) **then**
    return h^*
else
    return h
end if

The code selects the best candidate before applying the statistical gate. This ordering differs from testing every candidate and selecting the best among passing candidates. Reusing a stored parent measurement saves inference calls, but makes successive decisions share measurement noise. Neither detail is removed when interpreting the historical results.

The original blind pilot's program changes, recorded scores, costs, and paired decisions are summarized in Table 5. These measurements describe one selected trajectory, not averages over repeated searches.

B Analytic power and observed acceptance resolution

We distinguish the smallest *observed* gain that can pass the gate from the probability of detecting a true gain. For a fixed break-to-fix ratio ρ, define

$$r_\rho(n) = \min\Big\{ \frac{c-b}{n} : b = \mathrm{round}(\rho c),\ 0 \le b < c,\ b + c \le n, \\ \frac{c-b}{n} > 0.03,\ p_{\mathrm{exact}}(b, c) < 0.05 \Big\}. \tag{4}$$

With no breaks, the boundary is $6/40 = 0.150$ at $n = 40$ and $7/200 = 0.035$ at $n = 200$. This is an acceptance-resolution calculation, not a minimum detectable effect at specified power.

Table 7: Settings and their scope. Planned replication and measured pilot configurations are distinguished. Actual fixed-code repetition counts appear in Table 4.

Setting	Value and scope
Selection examples / candidates per generation	40 / 2 in the historical search
Generations	3 in the pilot; 5 per planned replication run
Acceptance	gain > 0.03; exact two-sided $p < 0.05$; positive paired direction
Search objective	accuracy ($\lambda = 0$); inference cost reported separately
Smoke examples / error-rate cap	4 / 20% in historical validity checks
Smoke cost / exception repair	at most $3\times$ parent / up to 2 passes
Mutation sampling	temperature 0; frozen operator chain after pilot generation 1
Replication target	eight complete runs per arm; reached for blind at the 18 September snapshot, guided and random pending
Replication guard	4500 CU or eight hours per run
Stall monitoring	1800 seconds without progress triggers external handling
Matched-backend inference	one fixed served model per backend; 8192 generation tokens per call; temperature 0

For an explicit power example, suppose independent paired examples have discordance probability q and true accuracy difference δ. Then $D = b + c \sim \text{Binomial}(n, q)$ and $c \mid D \sim \text{Binomial}(D, (q + \delta)/(2q))$. Summing the probability mass satisfying Eq. (2) gives the single-comparison power. With $q = 0.30$ and $\delta = 0.10$, it is 0.133 at $n = 40$ and 0.703 at $n = 200$ (Figure 5). These are analytic values under an illustrative independent-item model, not estimates of the live search's power. They show why a small acceptance threshold must not be confused with reliable detection of a small effect.

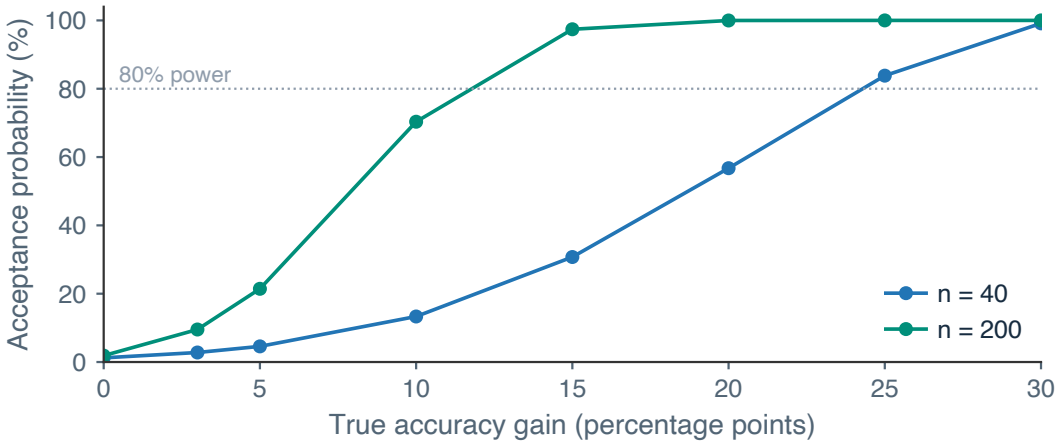


Figure 5: Single-comparison power under an explicitly specified model of independent paired examples with discordance probability $q = 0.30$. The curves sum binomial probabilities analytically; they are not measured power in the adaptive search. An observed gain can cross the gate's acceptance boundary even when the probability of detecting a true gain of that size is low.

For each $d = 0, \ldots, n$ and $f = 0, \ldots, d$, the power calculation weights the acceptance indicator by

$$\binom{n}{d} q^d (1-q)^{n-d} \binom{d}{f} \left(\frac{q+\delta}{2q}\right)^f \left(\frac{q-\delta}{2q}\right)^{d-f}. \tag{5}$$

Here f is the number of fixes and $d - f$ the number of breaks. The indicator requires $(2f - d)/n > 0.03$, $f > d - f$, and a two-sided exact $p < 0.05$. This specifies a full distribution under the alternative, unlike a minimum significant observed difference. Under $q = 0.30$, a true 15-percentage-point gain has power 0.307 at $n = 40$ and 0.974 at $n = 200$. The assumed q is illustrative; correlated screenshots, endpoint drift, and candidate selection invalidate a direct interpretation as the experiment's achieved power.

The historical same-code audit forms 36 seed comparisons from nine executions, plus one pair for each of two other repeated programs. Their orientation is inherited from the audit's input order. Each execution appears in several pairs, so increasing the number of pairs combinatorially does not increase independent measurement information in the same way. This also limits treating 2/38 as a binomial false-acceptance rate. The original source traces and pair identities are retained to permit alternative analyses.

C SCREENSPOT BREAKDOWN AND SAMPLE IDENTITY

Historical transfer comparisons use exact paired McNemar tests, keyed on stored sample identity and checked against instruction and target box. The 60-example comparison has 46 fixes and one break ($p = 6.8 \times 10^{-13}$); the full benchmark has 888 fixes and 22 breaks ($p = 2.0 \times 10^{-230}$). Per-repeat SynthGUI-test comparisons have $p \leq 2.2 \times 10^{-4}$, and both dark-theme comparisons have $p < 10^{-4}$.

On the historical full ScreenSpot evaluation, the champion hits 309/334 desktop targets (92.51%), 456/502 mobile targets (90.84%), and 369/436 web targets (84.63%). By target type it hits 657/697 text targets (94.26%) and 477/575 icon targets (82.96%). All 1,272 examples have valid execution measurements. The overall result is 1134/1272 (89.15%), at 3.097 historical CU per example.

The 60-example comparison is sampled using a fixed stored ordering. Its correct pair identity combines evaluation chunk and within-chunk index, then checks the instruction and target box. A previous instruction-only key merged two examples with matching instruction strings and produced 59 pairs. The corrected 60-pair comparison yields 46 fixes, one regression, 12 common hits, and one common miss. Only the corrected comparison is used in the paper. The new matched-backend study reuses a separately frozen 300-ID list and hashes both that list and image pixels; the saved IDs are authoritative because the older sampling script used Python hash randomization in its stratum seed.

The qualitative panel uses four frozen case identities: dataset indices 50 and 275 (Windows and macOS recoveries), 907 (the sole regression), and 785 (the sole common miss). These illustrate outcome categories rather than estimate their frequencies. The regression inset magnifies the original image around the champion prediction and the target's left boundary; the reported 4.08-pixel gap is measured in source-image coordinates.

D FEEDBACK CONTROLS AND DATED REPLICATION SNAPSHOT

In the original three-generation pilot, guided mutation accepts no candidate, whereas blind mutation accepts two. Guided failures include code errors, infrastructure-invalid evaluations, and an expensive multi-call candidate. Allowing additional repair in a separate guided control produces one measurable candidate with zero accuracy; this single control cannot establish whether repair budget explains guided performance.

The first pilot generation also uses different operators across arms. A matched-operator blind control accepts at 0.750 dev accuracy (13 fixes, two breaks, $p = 0.0074$), and a random-context control accepts a first-generation candidate at 0.975 while its sibling scores zero: the initial seed admits large improvements under several mutation contexts. This does not establish their relative search efficiency or a harmful effect of diagnosis.

The replication snapshot in Table 8 is more informative about that boundary: every completed guided run has an acceptance (11/11 including terminated runs), versus six of eight blind runs and all eight terminated random runs; a two-sided Fisher test on any-accept between completed guided and blind runs gives $p = 0.47$. These counts establish neither equivalence nor superiority—completion is unequal, one guided run remains active, and any-accept is only one property of a search. Terminal dev accuracy of completed runs (guided 0.94 ± 0.11, $n = 6$; blind 0.79 ± 0.26, $n = 8$; random 0.99 ± 0.01, $n = 4$; mean $\pm$ sample SD) is a selection-end measurement on the dev split, not an independent held-out result.

E OPERATOR AND INFRASTRUCTURE OBSERVATIONS

The probe records whether each operator returned a usable program under the tested payload and service limits. Failure labels describe the observed transport or output behavior. They do not imply that the model could not generate a valid program under another budget or endpoint configuration.

Table 8: Dated replication snapshot (18 September 2026), not the final pre-registered comparison. "Complete" means all five generations recorded; terminated runs also include budget/time aborts. Entries are runs with at least one acceptance / runs in the stated group. The active guided run is excluded from the terminated column. No terminal-accuracy means are treated as independent held-out results.

Protocol	Started	Complete	All terminated	Active
Guided	12	6/6	11/11	1
Blind	8	6/8	6/8	0
Random	8	4/4	8/8	0

Table 9: Historical operator probe under one approximately 5.2k-character payload and gateway budget. Usability refers to a parseable program passing the tested code filter. These single-configuration observations are not a general model ranking.

Served alias	Usable	Wall time (s)	Observed failure
deepseek_flash	yes	14	–
deepseek_pro	yes	57	–
ling_2_5_1t	yes	70	–
minimax_m2_7	no	66	thinking tags; truncation
qwen35_122b	no	88	incomplete code
kimi_code	no	104	empty visible content
glm_5_1	no	161	empty visible content
qwen35_397b	no	81	empty visible content
kimi_k2_6	no	154	watchdog cutoff
kimi_k3_stable	no	184	watchdog cutoff

F ARTIFACTS, CHECKS, AND DISCLOSURE OF INCOMPLETE EVIDENCE

The artifact layout separates source harnesses, model-call traces, lineage decisions, immutable manifests, and generated paper assets. A dated snapshot records source hashes for replication lineages and counts completed, terminated, and active runs independently. It does not relabel aborted runs as completed or use their omission to claim search reliability. Last-accepted dev accuracy is a selection measurement, not an independent terminal test score.

The new direct controls are implemented independently as one-call explicit-point and explicit-bbox interfaces. Offline checks verify normalized conversion, rejection of out-of-contract coordinates, and malformed-response handling before API evaluation. Their source hashes are frozen with the historical champion and seed. Model credentials are loaded from local configuration and are never placed in the public artifact manifest. The matched-backend client records token usage, call counts, errors, and endpoint identity separately from historical CU.

Figures are produced by deterministic scripts from saved data. Qualitative examples retain both positive and negative outcome categories; crops and inset locations are visual aids and do not change measured predictions or target boxes. The search diagram is authored in TikZ. The teaser, coordinate-contract diagram, measurement figure, and qualitative case layout are editable draw.io diagrams. Numerical marks are generated from frozen records; the qualitative figure embeds original screenshots and lossless crops with vector annotations at the recorded coordinates. AI assisted the revision and code used for these assets, as disclosed in the main paper. No generated image is used as experimental evidence.

The revised draft intentionally distinguishes completed measurements from planned experiments. The eight-complete-runs-per-arm replication and broader multi-backend search are not claimed complete. The new matched-backend evaluation and disjoint-pair null validation are complete; they evaluate fixed programs, not new searches. A submission package must include the final frozen artifacts and update the replication snapshot if further runs finish; preliminary results must remain identifiable as such.

Table 10: Selected operational incidents and the measurements they threaten.

Incident	Measurement risk	Recorded mitigation
Rate-limit storm	all-failed parent mistaken for accuracy zero	error-rate validity checks
Hanging response (9885 s)	elapsed time and search budget become misleading	external wall-clock watchdog
Invalid generated code	exceptions mistaken for grounding misses	smoke test and exception repair
Expensive multi-call program	full evaluation consumes disproportionate budget	smoke cost check
Generator import attempt	candidate may access benchmark construction	AST import filter
High concurrency slowdown	unequal completed work under a time budget	explicit throughput and failure logs

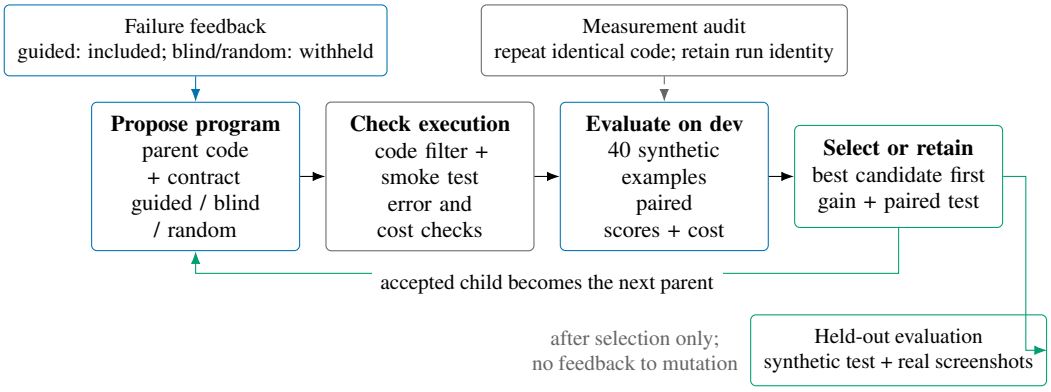


Figure 6: The search and its information boundaries. The mutation context changes across guided, blind, and random protocols. Execution checks precede selection on the synthetic dev set. Held-out screenshots are used only after a program is chosen. Identical-code re-evaluation audits the measurement process; it is not an additional source of labels for blind mutation.

G Search diagram

H Contract-violation audit of frozen matched-backend responses

A post-hoc inspection of the frozen Qwen responses finds 37 direct-point control outputs outside $[0, 1]$; the direct-bbox control has 74 such outputs plus 27 malformed or missing-contract responses. The strict parsers reject these; they remain misses in operational and paired scores, distinct from the 29 empty-response/transport failures across the four arms. We do not rescore them after observing results. The point baseline's advantage therefore holds with its recorded contract failures, while the bbox result cannot isolate a pure box-versus-point effect.

I Matched-backend paired statistics

Primary comparisons use two-sided exact McNemar tests on common error-free items, with Holm adjustment across champion versus direct point and champion versus direct bbox separately within each backend. On Qwen, the champion fixes 30 and breaks 77 versus direct point on 289 items (-16.26 paired percentage points; Holm $p = 1.26 \times 10^{-5}$). Against direct bbox, it fixes 58 and breaks 59 on 289 items (-0.35 points; Holm $p = 1$). On Kimi, the champion fixes five and breaks 11 versus direct point on 282 items (-2.13 points; Holm $p = 0.420$), and fixes six and breaks seven versus direct bbox on 279 items (-0.36 points; Holm $p = 1$). Operational accuracy in Table 2 instead uses all 300 items per arm, counting execution failures as misses. The operational differences between champion and direct point are -16.33 points on Qwen and -4.00 on Kimi.

J HISTORICAL SHARED-EXECUTION AUDIT

Nine valid evaluations of identical seed code on the same dev examples span 0.350–0.550 accuracy (mean 0.475, sample standard deviation 0.058). A tenth evaluation preceding an empty-response fix is excluded as an infrastructure artifact. Between repeats, 7–35 point predictions and up to 14 hit/miss outcomes differ despite temperature zero.

The audit contains 36 comparisons from these nine seed runs plus two from repeated child programs (13 executions in total). Gain-only accepts 17/38; a bounded-net-wins rule (at least two net fixes and at most five breaks) accepts 16/38; ValidGate accepts 2/38. Both paired-gate acceptances compare the lowest seed measurement with higher same-code measurements (9/1 and 6/0 fixes/breaks). These are false program-improvement decisions in the recorded comparisons.

The historical audit implementation checks gain and exact p without a separate positive-direction clause. Both accepted pairs have more fixes than breaks, so adding the direction check in Eq. (2) leaves the count in Table 6 unchanged.

The counts do not estimate an independent 5.3% false-positive rate: comparisons share runs, orientation follows stored audit order, and the audit helped motivate the rule. Reapplying historical rules to stored lineage decisions does not establish a causal benefit for final search quality.

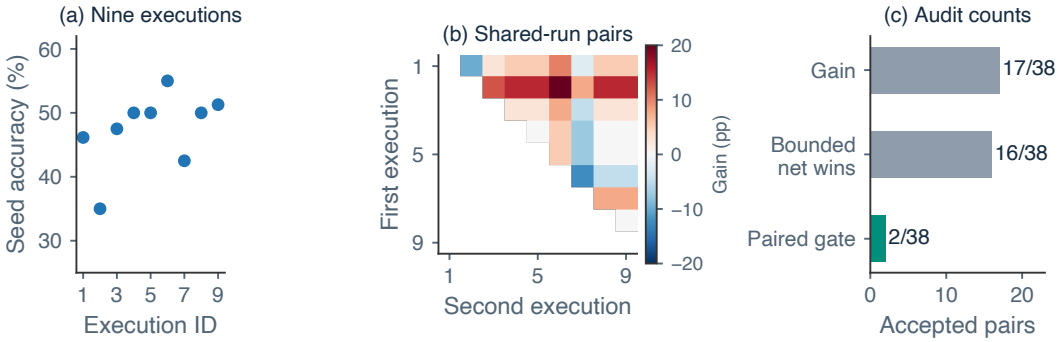


Figure 7: Historical same-code audit. Left: nine seed execution accuracies. Center: the signed accuracy differences for their 36 oriented pairs, showing explicitly that comparisons share runs. Right: descriptive acceptance counts over those 36 pairs plus two pairs from other repeated programs. The 38 pairs are dependent, so no independent-binomial confidence interval or nominal-FPR reference line is shown.

New validation protocol and execution accounting. The matched-backend studies each contain 1,200 frozen program–item jobs. Kimi's first 37 jobs at concurrency 16 recorded 12 API errors; after separate transport probes, only missing jobs resumed at concurrency four. All initial errors remain in the analysis (35 across the four completed arms). The fresh Qwen validation contains 1,600 same-code null jobs and 2,000 dev200 jobs. All 240 rendered input images, program hashes, preassigned task IDs, and returned records are saved. The five dev200 seed repeats hit 82, 91, 86, 82, and 82 targets with 13, 15, 9, 11, and 10 errors; champion repeats hit 125, 128, 117, 130, and 117 with 3, 2, 4, 4, and 3 errors. Each denominator is 200, including errors as misses. No recorded failure is replaced by a later successful retry.