Beyond Chain Selection: Compiling Proposal Archives into Frozen Agent Harnesses

Anonymous authors
Paper under double-blind review

Abstract

Agent harness evolution couples two distinct operations: proposing useful edits and selecting edits that generalize. We study whether useful proposal content can be recovered even when chain-level selection fails. We introduce cross-chain proposal compilation, which constructs a single frozen harness from logged edits with no additional model calls at compile time. We instantiate this idea in two forms: prose compilation assembles proposal text into instructions, and typed compilation selects recurrent, executable mechanisms with development-ranked implementations. On a primary pool of 1,007 BFCL v4 tasks, union prose, recurrent prose, and typed consensus improve paired held-out accuracy by +2.29, +2.02, and +1.65 percentage points. All three paired intervals exclude zero. A fresh five-arm follow-up on 400 additional tasks finds smaller effects whose intervals include zero, indicating limited category transfer. In a separate held-out comparison, single-chain finalists underperform the baseline despite positive development gains. These results support proposal-level reuse beyond chain-level selection, with bounded evidence for transfer and inference cost isolated as a separate outcome.

1 Introduction

An LLM agent's harness specifies its prompts, tool interfaces, memory, parsing, and control flow. Changing this scaffolding can improve performance without updating model weights (Yang et al., 2024; Lee et al., 2026).

Recent systems automate these changes: a meta-model proposes edits, evaluates candidates, and retains promising versions (Hu et al., 2024; Zhang et al., 2025b; Agrawal et al., 2025; Lee et al., 2026). The resulting evolution chain couples two distinct operations: generating useful edits and deciding which of them generalize.

These two operations need not succeed together. Acceptance decisions are made under sampling noise, on development tasks that the chain may overfit, and over edits that may never execute. A chain can therefore accumulate accepted edits while its measured gains reflect noise or development fit rather than transferable improvement.

We observe this divergence directly: even a behaviourally unchanged harness can receive an accepted edit and a positive measured gain. Single-chain finalists underperform the baseline on held-out tasks despite positive development gains. This motivates our central question: can useful proposal content be recovered even when chain-level selection fails to generalize?

We introduce cross-chain proposal compilation: reuse logged proposals to construct one frozen harness. We instantiate it in two forms. Prose compilation extracts proposal text and assembles it into instructions, using either a broad union or recurrent content. Typed compilation, implemented as consensus, selects recurrent, executable mechanisms and copies development-ranked implementations. Each compiled artifact runs as one harness; compilation itself makes no additional model calls. Generating the input chains and executing the artifact still incur costs (Figure 1).

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

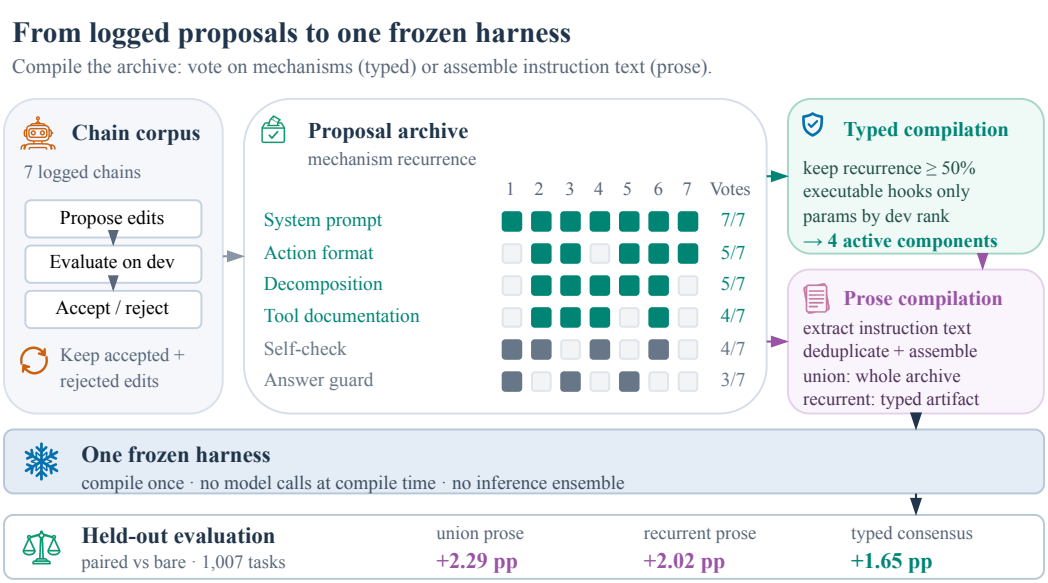


Figure 1: From logged proposals to one frozen harness. Chains propose, evaluate, and accept or reject edits; the archive retains every proposal. Typed compilation keeps mechanisms that recur in at least 50% of chains on executable hooks, copying development-ranked parameters. Prose compilation assembles instruction text from the whole archive (union) or the typed artifact (recurrent). Bottom right: paired primary-pool gains over bare; single-chain failures come from a separate comparison (§5.4).

Both prose variants and typed consensus improve paired held-out accuracy on the primary pool. A fresh expansion on further categories finds smaller effects whose intervals include zero.

Our contributions are:

(1) A framework for proposal-level reuse. We formalize cross-chain proposal compilation and give two reproducible instantiations: prose compilation and typed consensus (§3).
(2) Held-out evidence that proposal archives contain recoverable value. On 1,007 BFCL v4 tasks, all three compiled variants improve paired accuracy over the bare harness (§5).
(3) A reliability analysis of chain-level selection. Development gains, acceptance decisions, and held-out performance can diverge; we quantify evaluation variability, inactive edits, and inference cost (§6).

2 Related work

Automated improvement of agents. Prompt and LM-program optimization searches instructions or modular programs (Zhou et al., 2022; Yang et al., 2023; Pryzant et al., 2023; Fernando et al., 2023; Guo et al., 2023; Khattab et al., 2023; Opsahl-Ong et al., 2024; Yüksekgönül et al., 2024). Other systems update context, memory, or experience (Shinn et al., 2023; Zhao et al., 2024; Madaan et al., 2023; Wang et al., 2024; 2023; Zhang et al., 2025c). Workflow and program search expand the editable object (Hu et al., 2024; Zhang et al., 2024; Zhuge et al., 2024; Shang et al., 2024; Zhang et al., 2025a; Zelikman et al., 2023; Yin et al., 2024; Zhang et al., 2025b). Harness-specific work searches code, co-evolves weights and harnesses, or uses self-preference (Lee et al., 2026; Luo et al., 2026; Pan et al., 2026). Related systems compile context into skills or specialize by task family (Wei et al., 2026; Ning et al., 2026; Huang et al., 2026b;a). Proposal compilation complements these searches by assembling logged content into one harness beyond the chain's final acceptance history.

Algorithm 1: Cross-chain proposal compilation. Deterministic construction; no model calls at compile time.

Input: archive $\mathcal{A}$, source scope $\mathcal{S}$, selection rule $\mathcal{R}$, assembly rule $\mathcal{B}$.

1. Filter proposals by scope $\mathcal{S}$; extract payloads (typed mechanism identities or prose instruction text).
2. Select under $\mathcal{R}$: recurrence and coverage (typed), or deduplication and length control (prose).
3. Assemble the survivors under $\mathcal{B}$ into a single harness; freeze it.
4. Evaluate the frozen harness on held-out tasks.

Evaluation and selection. Error bars and leakage controls are central to reliable empirical comparisons (Miller, 2024; Kapoor & Narayanan, 2023). Recent harness studies distinguish the ability to write updates from the ability to benefit from them (Wu et al., 2026; Lin et al., 2026). We evaluate prose and typed proposal compilation alongside repeated-measurement variability and runtime coverage. The typed surface makes reconstruction and static analysis tractable but restricts the search space; the prose variants recover textual content without processor-level execution structure.

3 Cross-chain proposal compilation

Archive and compiler. Let $\mathcal{A} = \{(e_i, c_i, s_i)\}_{i=1}^{N}$ be an archive of proposed edits, where e_i is the edit payload, c_i its chain provenance, and s_i its selection status. A proposal compiler $\mathcal{C}$ maps $\mathcal{A}$ to a single frozen harness $H^* = \mathcal{C}(\mathcal{A})$ under a deterministic construction rule, with no model calls at compile time. We study two instantiations of this abstraction. Typed compilation selects mechanisms that recur across a BFCL chain population, filters them by runtime coverage, and copies development-ranked implementations. Prose compilation extracts proposal text and assembles deduplicated instructions: union prose reads the broader archive, and recurrent prose derives from the typed artifact. Algorithm 1 states the common recipe.

3.1 Harnesses and evolution chains

A harness H is a JSON specification executed by a frozen runtime around a frozen base model M. Processors attach to 8 lifecycle hooks and belong to 13 kinds, each with a parameter schema and legal hooks. An edit applies one of 6 operators (Appendix A.3). A chain is a sequence $H_0 \to \cdots \to H_T$ produced by repeated proposal, evaluation, and acceptance. The population contains chains generated under a common experimental budget, with different seeds and meta-model families. Separate generation runs need not be statistically independent: chains can share an initialization or proposal model.

Frozen execution and logged selection. The interpreter records its hash and cannot be edited by the meta-model. EvoLog records proposed edits, parameters, rationales, gate decisions, transcripts, and harness versions. Static checks validate types and development-specific literals; a smoke test checks execution. The see-saw gate additionally limits task-level regressions on the development window. Conditional on these checks, the baseline acceptance rule is

$$\mathrm{acc}(H_{t+1}) = [\hat{\Delta}_{\mathrm{dev}} > 0] \wedge [\mathrm{reg}(H_{t+1}, H_t) \leq \tau]. \tag{1}$$

Here reg counts regressions, and τ is calibrated from the measured flip rate. The default rule has no token-cost constraint. A separate ablation imposes a $1.10\times$ token cap (§6.3). Adaptive re-evaluation of tied tasks is used in 42.5% of 40 decisions. The corpus records 113 proposals and 40 decisions (Appendix D.1).

3.2 Typed compilation: mechanisms and parameters

Let P_c be the set of mechanisms proposed anywhere in chain c. A mechanism is identified by its operator, processor kind, and hook, written *op*:*kind*@*hook*. Its recurrence is

$$\hat{r}_m = \frac{1}{C} \sum_{c=1}^{C} \mathbf{1}[m \in P_c]. \tag{2}$$

Repeated proposals within one chain contribute only one vote. Consensus applies the following rules to the logged population:

F1. Mechanism identity. Count coarse mechanisms rather than parameter-specific fingerprints.
F2. Recurrence. Retain mechanisms proposed by at least 50% of chains.
F3. Coverage. Remove components whose hooks or kinds cannot execute in the target environment.
F4. Parameters. For each retained mechanism, copy its implementation from the highest-final-dev chain that proposed it.

The construction excludes scalar parameter edits and produces 4 active components. Appendix C.5 details ordering and component caps.

The artifact is built from 7 chains on BFCL `live_multiple` development tasks and frozen before held-out evaluation. Rebuilding on the policy-clean 4 chains produces a behaviourally identical artifact (Appendix A.2).

Recurrence selects mechanisms, but parameter ranking and later proposals still depend on development evaluation. Generating the source population remains part of construction cost.

3.3 Prose compilation: proposal text as instructions

Prose compilation collects string-valued instruction fields from proposed edits, whether accepted or rejected. It deduplicates and sorts the text blocks, then joins them into one system-prompt component. Assembly is deterministic and model-free: it preserves the source text without resolving conflicts between instructions.

Union and recurrent variants. Union prose scans the archived edit logs across environments and supplementary runs, then retains a character-capped prefix of the sorted blocks without acceptance or recurrence filtering. Recurrent prose applies the same text extraction to the selected typed consensus artifact. It therefore inherits that artifact's mechanism and development-ranked parameter choices, while discarding its processor-level execution structure. The evaluated artifacts and exact extraction rules are recorded in Appendix C.2.

Length control. A seeded subset of union text forms the matched-prose control. Its character budget approximates the typed artifact's incremental token cost using archived measurements; it does not enforce equal realized inference tokens.

4 Evaluation protocol

Tasks, models, and baselines. Our main evaluation uses BFCL v4 tool calling with AST-subset matching (Patil et al., 2023; 2026). ToolOrbit provides a synthetic, state-assertion environment for diagnosing evolution behaviour. MGSM-en is a saturation probe; verified multi-turn integration is reported separately (Appendices A.5 and C.6). The headline task model is Qwen3.8-27B, with weights fixed within each comparison. Two meta-model families generate edits; deterministic checkers, rather than LLM judges, score task outputs. The seed library contains bare and transcriptions of public harnesses, pinned to upstream revisions (Appendix E.2). The public-harness controls evaluate these specifications rather than the original full systems.

Table 1: Compilation variants and reuse controls on the primary pool. All effects are paired with the corresponding archived bare evaluation on 1,007 tasks. Intervals describe mean sampled accuracy; they do not test superiority between variants. Token ratios use task-weighted primary-stratum accounting.

Harness	Δ (pp)	Paired 95% CI	Tokens / bare
Union prose	+2.29	[+1.06, +3.54]	2.02
Recurrent prose	+2.02	[+0.86, +3.17]	1.14
Typed consensus	+1.65	[+0.58, +2.74]	1.22
Matched prose	+0.73	[-0.56, +2.03]	1.31
Accepted checklist	+0.03	[-0.82, +0.86]	1.00
Proposed checklist	-4.37	[-5.75, -3.03]	1.17

Held-out pools and comparisons. A stratum is a disjoint task slice evaluated under one fixed protocol. Clean strata exclude tasks used by chains, gates, probes, or development evaluation. The primary pool contains 1,007 tasks; the expanded pool contains 1,407 tasks and adds further BFCL categories. The abstention category is reported separately, and the exploratory development-side stratum is excluded from both pools. All effect sizes compare arms on the same tasks within a stratum. The single-chain comparison uses a separate 300-task evaluation with its own baseline, reported apart from the pooled scores. Registration timing and the complete inclusion policy appear in Appendices A.1 and A.2.

Effect sizes and inference. For each task, accuracy is the fraction of successful samples; k and repeated evaluations are fixed within each comparison. We average paired differences over tasks, weighting strata by task count. Confidence intervals preserve task pairing. The typed evaluation resamples within strata; the archived prose evaluation resamples the pooled task list. McNemar tests use a binarized endpoint: a task succeeds when its sampled success fraction is at least one half. Exact two-sided tests on this endpoint need not agree with bootstrap intervals on mean sampled accuracy. For typed consensus, the registered decision rule requires both $p < 0.05$ and a confidence interval excluding zero. Archived prose evaluations cover the primary pool; archived expanded and abstention results cover typed consensus. A separate follow-up evaluates all compilation variants with a fresh common baseline (§5.2). Token costs include inference tokens per task; construction and evaluation accounting are reported separately in the supplement.

5 Held-out results

5.1 Prose and typed compilation improve pooled accuracy

Both prose variants and typed consensus improve primary-pool accuracy (Table 1); their paired intervals exclude zero. Effects use archived paired baselines and do not test superiority between variants. Figure 2 reports typed consensus's broader evaluation.

Broader evaluation of typed consensus. The archived expanded estimate is +1.68 pp, with positive point estimates in 8/8 evaluated strata. Individual intervals often cross zero; positive direction does not imply significance in every stratum. Consensus incurs 1.18–1.38× the baseline tokens across expanded-pool strata. Abstention is evaluated separately and yields +5.62 pp, with exact McNemar $p = 0.004$. These additional results concern typed compilation only.

5.2 Fresh expansion yields smaller, unresolved effects

We froze the same artifacts and evaluated five arms on 400 clean tasks from further BFCL categories, with $k = 3$. The supplementary protocol was specified before these new evaluations; it is separate from the original pre-registration (Appendix C.3). All arms use a fresh common baseline, and no prompt is retuned. Table 2 reports every arm and category. All

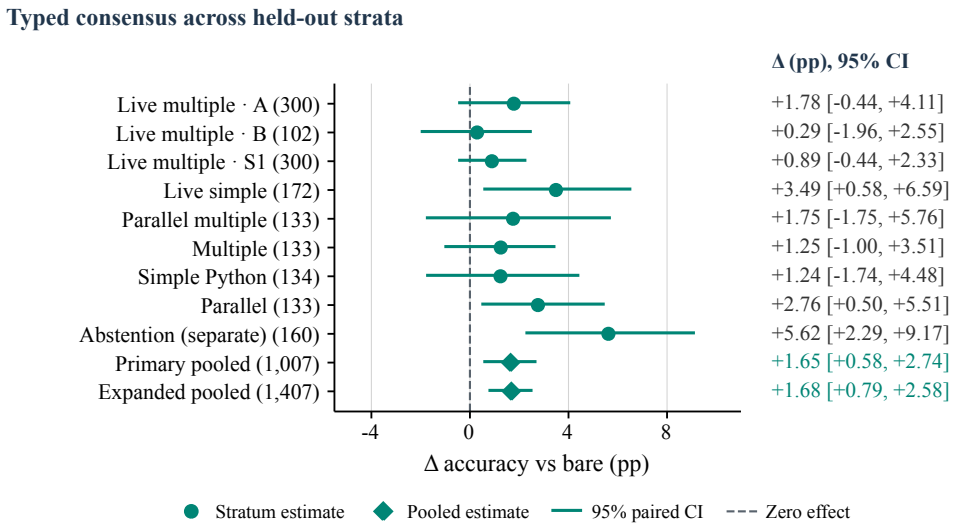


Figure 2: Archived typed-consensus evaluation across held-out strata. Bars show task-level paired bootstrap 95% intervals; parentheses give task counts. Diamonds denote nested pooled estimates; abstention is a separate evaluation. Table 1 compares compilation variants on the primary pool.

Table 2: Frozen-artifact transfer with a fresh common baseline. Multiple, Simple Python, and Parallel contain 400 tasks in total ($k = 3$). Bare cells give accuracy; other category cells give paired differences (pp). Pooled intervals resample tasks within strata; token ratios measure inference cost.

Harness	Multiple	Python	Parallel	Pooled Δ (pp) 95% CI	Tokens / bare
bare (accuracy)	91.73%	92.54%	91.48%	---	1.00
Union prose	+1.25	-1.24	+1.50	+0.50 [−0.92, +2.00]	2.75
Recurrent prose	+1.00	-0.50	+1.50	+0.67 [−0.58, +2.00]	1.23
Typed consensus	-0.50	+0.25	+1.50	+0.42 [−1.00, +1.83]	1.32
Matched prose	-0.75	-3.23	+0.25	−1.25 [−2.92, +0.42]	1.52

compiled-artifact intervals include zero, while inference overhead remains. Transfer to these further categories is therefore limited.

5.3 What do selection and representation contribute?

The pre-registered recurrent-minus-union prose contrast is -0.27 pp, with an interval spanning zero (Appendix C.2). Source scope, content, selection, and execution structure differ jointly across these constructions, so the point estimates do not rank the variants.

The proposed-edit checklist has a negative effect on the primary pool, while the accepted-edit checklist is near zero: not every reuse procedure yields a gain. Both constructions are reconstructible and incur inference overhead (Figure 4); typed components additionally support static coverage analysis.

5.4 Single-chain finalists degrade in a separate comparison

Both evolved finalists score below bare on the 300-task held-out comparison (Table 3). They also consume more inference tokens. Meanwhile, the development logs report a mean gain of +6.67 pp. ToolOrbit exhibits the same direction reversal between development and held-out evaluation (Appendix C.1). This resembles the selection--generalization divergence studied in reward optimization (Gao et al., 2022), and concerns these chains, task models, and evaluation windows rather than harness evolution in general.

Table 3: Single-chain finalists and checklist controls. BFCL `live_multiple`, n=300, k=3. All rows use the same task list and baseline evaluation. The three behaviourally equivalent aggregation arms are reported separately in Appendix C.1.

Harness	Accuracy (%)	Δ (pp), 95% CI	Tokens/task
bare	80.33	---	1431
Accepted checklist	79.44	−0.89 [−2.44, +0.67]	1430
Evolved, bare seed	75.78	−4.56 [−7.00, −2.22]	1577
Evolved, SWE seed	75.67	−4.67 [−6.89, −2.56]	1590
Proposed checklist	73.44	−6.89 [−9.78, −4.22]	1659

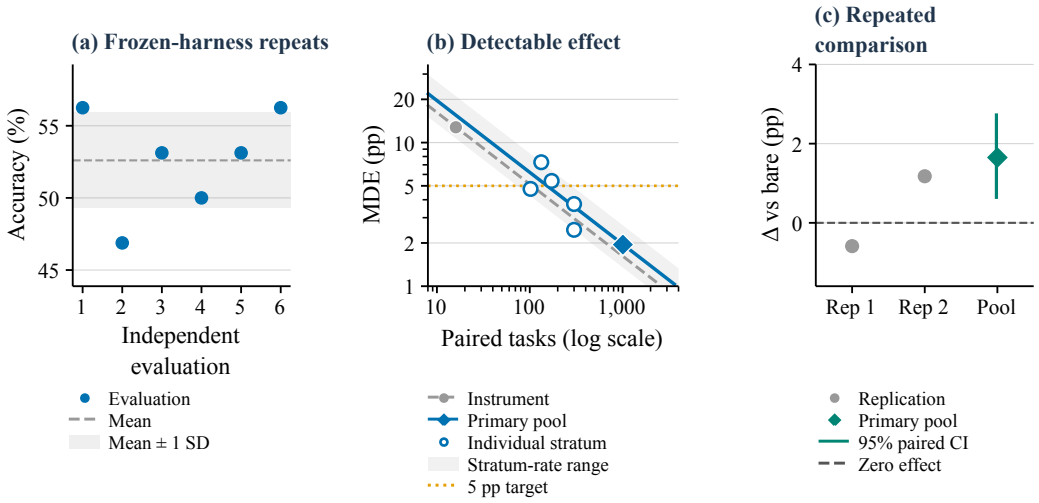


Figure 3: Measurement variability limits per-edit selection. (a) Repeated evaluation of one frozen ToolOrbit harness (n=16, k=2). (b) Approximate binary-endpoint MDE from instrument, stratum, and pooled discordance rates. Curves and markers are planning estimates, not measured accuracy; shading spans stratum rates. (c) Repeated stratum-B effects and the primary-pool estimate with its paired bootstrap interval. Panel legends identify every line, marker, and shaded region.

Task-dependent harness effects. Public-harness transcriptions show task-dependent effects: an interface that helps abstention can hurt parallel tool calling, and near-ceiling categories discriminate little (Appendix E.2.2). These are controlled harness specifications, not the original full agent systems.

6 Analysis of selection reliability

6.1 Evaluation variability and statistical resolution

Repeated evaluation of one frozen harness measures variability without any edit. On ToolOrbit, 6 independent evaluations yield SD 3.3 pp and range 9.4 pp (Figure 3a). On the BFCL development window, chains with the same seed specification recorded markedly different baseline scores (Appendix A.2). Such variability is relevant to a loop that accepts candidates based on small positive differences. It is not a universal noise constant: it depends on tasks, sampling, and the evaluation endpoint.

Near the null, the approximate binary-endpoint minimum detectable effect is $2.802\sqrt{q/n}$ for discordance probability q and task count n. At $n = 12$, the measured instrument discordance implies an MDE of 14.8 pp. This planning approximation excludes adaptive selection uncertainty; it is not a confidence interval for mean sampled accuracy.

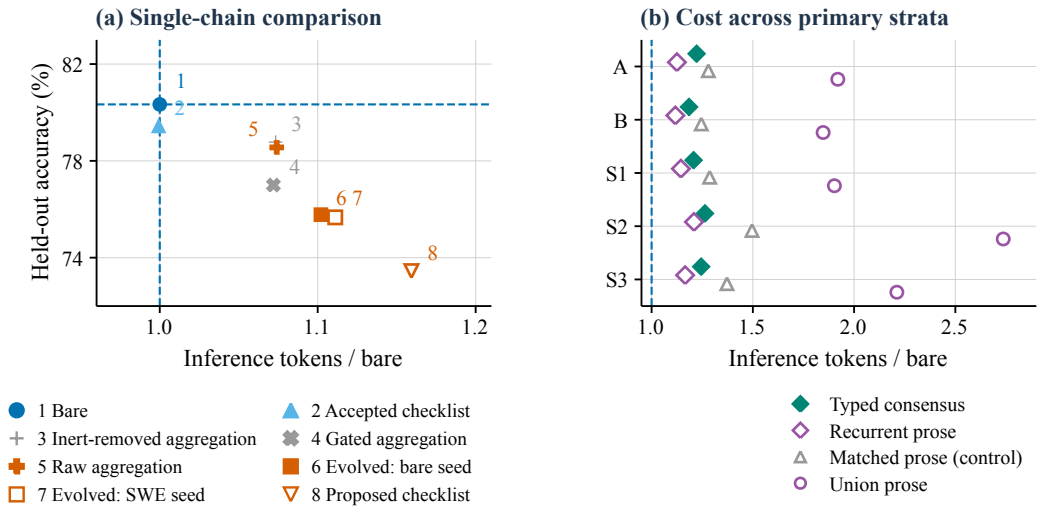


Figure 4: Inference cost is a separate outcome. Left: accuracy and cost for all arms in the 300-task single-chain comparison; the upper left is preferable. Right: token ratios for prose and typed compilation, plus the matched-prose control, within each primary-pool stratum. The dashed reference denotes bare cost. These ratios measure inference only, not the cost of generating the proposal corpus.

Two further controls expose evaluation variability. First, single-step execution makes 3 aggregation arms behaviourally identical despite different serialized components. Their observed scores vary by SD 0.8 pp. Second, repeating the same comparison on stratum B changes the estimated effect by 1.76 pp (Figure 3c). These observations motivate paired replication and explicit uncertainty, rather than interpreting each accepted edit as established improvement.

6.2 What recurs across chains?

Proposed mechanisms recur more often than accepted mechanisms: their public-corpus Jaccard similarities are 0.441 and 0.173, respectively. Parameter-specific fingerprints agree less often, motivating mechanism-level rather than exact-edit voting. The proposal corpus contains 20 distinct mechanisms, including 12 singletons and 1 unanimous mechanism. At the chosen threshold, 6 mechanisms pass the recurrence rule. One targets an inactive hook; another is a non-component parameter edit. The released construction yields 4 active components (Appendix C.5).

All surviving components were proposed by chains from both meta-model families and both seed harnesses. This checks factor coverage, not vote independence or each component's causal benefit. Final-harness similarity remains higher within a seed than across seeds (0.688 versus 0.198). Only system_prompt is universal, so the population does not converge to one common final harness.

Unvoted typed aggregation performs worse, while unanimity leaves only 1 mechanism. Corpus exclusions and sensitivity results are reported in Appendix A.2. Table 4 shows that proposal similarity exceeds acceptance similarity under both BFCL inclusion policies and on ToolOrbit. Fine-grained agreement is weaker in both environments, supporting mechanism-level selection as a design choice.

6.3 Inactive components and inference cost

Static coverage analysis identifies components that cannot execute in a given environment. Among type-passing edits, 12.6% are inactive; 3/17 valid chains end behaviourally identical

Table 4: Proposal recurrence across environments and inclusion policies. C counts chains. The primary BFCL corpus uses the main inclusion policy; the relaxed corpus allows noise-excluded chains. These are nested analysis sets, not separate experiments. ToolOrbit uses a single seed, so across-seed similarity is undefined. Entries are descriptive means over chain pairs; the pairs are dependent.

Statistic	BFCL primary $C = 4$	BFCL relaxed $C = 6$	ToolOrbit $C = 4$
Proposed mechanisms: Jaccard	0.441	0.361	0.467
Accepted mechanisms: Jaccard	0.173	0.178	0.120
Proposed fingerprints: Jaccard	0.088	0.087	0.050
Accepted fingerprints: Jaccard	0.000	0.021	0.000
Final similarity: within seed	0.688	0.596	0.325
Final similarity: across seeds	0.198	0.246	---

Table 5: Acceptance-rule ablations expose the accuracy--cost trade-off. Each row compares its final harness with its own baseline. Default development gains average the corpus; its held-out range covers two finalists.

Rule	n/k	Dev Δ (pp)	Held-out Δ (pp)	Tokens / baseline
Default finalists	300/3	+6.67	-4.67 to -4.56	1.10–1.11
Cost cap (1.10×)	100/2	+0.00	-1.00	1.03
Deletion only	100/2	+0.00	+2.50	1.00

to their seeds. In particular, one of them accepted an edit and logged $\Delta = +0.008$. This is a concrete example of apparent improvement without changed execution. The original gates did not test reachability, whereas the coverage filter used by consensus does. The audit also finds processor kinds with no interpreter branch (Appendix D.1).

Component accumulation increases cost without guaranteeing held-out improvement (Figure 4). Table 5 compares the archived acceptance-rule ablations. The cost cap lowers measured token growth; its held-out estimate is underpowered for a non-inferiority claim. Deletion-only search accepts no deletions. These runs use different evaluation windows, so the comparison is descriptive.

7 Conclusion

Chain-level selection is unreliable, but proposal-level reuse is recoverable: this is the central finding of our study. Compiled from logged proposals, a single frozen harness improves held-out accuracy on the primary pool, even as the chains that produced those proposals fail to generalize. The broader lesson is methodological: evolution archives are first-class artifacts, and proposal generation and selection deserve separate evaluation, separate budgets, and separate reporting. Where an archive transfers, and at what cost, remains open; our expansion results bound, rather than settle, category transfer. Cross-chain compilation is a baseline that self-improving agent systems should report against.

AI use statement

Generative AI tools were used in the following capacities: (1) As objects of study. Every experiment in this paper consists of LLM calls: a frozen task model produces agent behaviour, and meta-models (the ``evolver'' and ``critic'') produce harness edits. These are the experimental apparatus; their outputs are the data, not claims of this paper. (2) For infrastructure. Coding assistants helped implement the experimental infrastructure. The resulting code paths are deterministic and were verified by the tests and checks described in the Reproducibility statement; all numerical claims are computed from logged outputs. (3) For prose. Language models assisted with drafting and editing English prose. All text was reviewed and rewritten by the authors; all scientific claims, all experimental design, all pre-registrations and all interpretations are the authors'. We take full responsibility for the content of this paper, including any text, code or artifact produced with the aid of generative AI. Every robustness check, every exclusion decision and every number in this paper is ours, not the models' (Appendix D.1).

Ethics statement

This work involves no human subjects, no personal data, and no new data collection from users. Evaluation uses public benchmark releases (BFCL v4 and MGSM, redistributed under their upstream licenses) and a synthetic environment we generated. It also uses internal tool-use traces that contain no user data. Our subject is a research methodology, so its risks are the ordinary ones of an audit paper. A negative result about harness self-evolution could be read as a claim that such systems are useless. That is not what we show. We show that a specific acceptance rule on a saturated single-turn benchmark does not beat its own seed, and that prose and typed proposal compilation yield positive pooled effects. The main dual-use consideration cuts the other way. The gates and coverage checks we release (§3) are detection instruments against reward-hacking-style self-modification. G4 in particular exists so that a self-modifying agent cannot claim credit for fixing a failure mode its environment does not contain. We disclose model access, compute and cost in the Reproducibility statement and Appendix B.1. We also release the EvoLog corpus, because an unverifiable claim in this area is, in our view, the more serious ethical problem.

Reproducibility statement

The supplement contains the rig and its frozen interpreter, the three environments with their verifiers, and the complete EvoLog corpus. It also contains the pre-registration documents, the analysis code, and the transcribed published harnesses with their upstream sources and revisions. That is everything needed to re-derive this paper. Six properties matter for a reader who wants to check us rather than trust us. (i) Generation, not transcription. No number in this paper is typed by hand. Every quantity is computed by deterministic code from the logged output of the measured chains and evaluations, and injected into the text. Every figure is drawn from the same data, so recomputing a measurement changes the paper. The supplement ships the resulting audit trail --- each quantity, its value and the log it came from --- as a machine-readable table rather than as prose. (ii) Pre-registration. The clean-pool evaluation protocol was registered before the new evaluations it specifies. The proposal corpus and the A/B evaluations already existed and informed the design. It fixes the arms, strata, inclusion and exclusion rule, test, decision rule, and the MDE achievable at the planned n (Appendix A.1, with its commit date). The exploratory `dev`-side stratum that would have flattered the result is excluded by that document, not by us. (iii) Logs. Every chain writes an EvoLog. It holds the configuration and interpreter hash, per-round proposals with their rationales, each gate's verdict and reason, and tie-break draws. It also holds per-episode transcripts and verifier output, per-call tokens and latency, and every intermediate harness version. 35 chains are released (15 with completed evolution rounds), together with a reconstruction check. That check recomputes any historical harness from its log and asserts it equals the archived specification. (iv) Environments and verifiers. BFCL v4 categories are scored by the official AST-subset matcher, extended to nested parameters, type coercion and optional-parameter omission (the extension and its unit checks are in Appendix A.5). MGSM is scored by exact-match numeric extraction over both answer channels the task

permits, saturation-probed over all 250 items before use (Appendix A.5). ToolOrbit is scored by state assertions with a verified oracle (90/90). Benchmark and upstream-harness sources, revisions and fetch dates are recorded in the supplement. (v) Models, cost and citations. No weights were trained. All models are served checkpoints, listed with role, family and access date in Appendix B.1. Task, meta and critic roles are assigned across different families to block self-distillation (the JSON-reliability screen behind that assignment is reported there). Sampling parameters are recorded per harness specification (temperature 0.3). Archived campaign spend: 110.4M tokens ($18.03 list-price equivalent), 5.3 wall-clock hours, zero GPU-hours. The supplementary follow-up reports its additional evaluation cost separately. Every experiment in this paper is reproducible by calling models. Every bibliographic entry was resolved against an external record, and only fields returned by that record were written: 56 entries verified and 10 rejected. The rejected entries include codenames of related systems we could not verify publicly and therefore do not cite. (vi) One command. A single pipeline in the supplement regenerates every number, table, figure and the compiled document from the released logs. The audit above is therefore re-runnable, not merely described.

Limitations

Scope and unresolved attribution. The headline evidence concerns single-turn BFCL tasks and one frozen task-model family. The strata share a benchmark ecosystem and are not independent demonstrations across application domains. MGSM is near saturation; multi-turn integration is verified but lacks a sufficiently powered comparison. The population is small, and chains sharing a seed or meta-model may make correlated proposals. The fresh expansion does not resolve small gains, and union prose draws from a broader archive than typed consensus. The prose arms were pre-registered as representation controls; we report their original contrasts, and the compilation reading of these arms is post hoc. The single-chain comparison uses a different task set and baseline, so its scores are not commensurable with the pooled estimates. We have not isolated each filter's contribution, established equivalence between the reuse variants, measured accuracy across intermediate recurrence thresholds, or identified the cause of the proposed-checklist degradation. The effect is modest and comes with inference overhead; practical value depends on the deployment budget. Per-stratum tests are unadjusted, and binary-endpoint power calculations do not quantify all uncertainty in sampled-accuracy estimates.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Ritu Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. arXiv (Cornell University), 2025. doi: 10.48550/arxiv.2507.19457. arXiv:2507.19457.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2309.16797. arXiv:2309.16797.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. arXiv (Cornell University), 2022. doi: 10.48550/arxiv.2210.10760. arXiv:2210.10760.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, Yujiu Yang, Tsinghua University, and M. Research. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. arXiv, 2023. doi: 10.48550/arxiv.2309.08532. arXiv:2309.08532.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2408.08435. arXiv:2408.08435.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. Envharness: Awakening static worlds

for agent learning. arXiv (Cornell University), 2026a. doi: 10.48550/arxiv.2608.19880. arXiv:2608.19880.

Haiyu Huang, Jiewei Lyu, Zhihan Jiang, Jinyang Liu, Xiao He, Tieying Zhang, Wu Xiang, and Michael R. Lyu. From general agents to rca experts: A self-evolving harness for root cause analysis. arXiv (Cornell University), 2026b. doi: 10.48550/arxiv.2608.25661. arXiv:2608.25661.

Sayash Kapoor and Arvind Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. Patterns, 2023. doi: 10.1016/j.patter.2023.100804.

O. Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2310.03714. arXiv:2310.03714.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2603.28052. arXiv:2603.28052.

Min Lin, Jun-Cheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bingtao He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, xiang zhang, Benoît Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Han Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2605.30621. arXiv:2605.30621.

Haochen Luo, Yi Huang, Sichun Luo, Fengyuan Liu, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. Harness-aware self-evolving: Co-evolving model weights, harness, and task solutions. arXiv, 2026. doi: 10.48550/arxiv.2607.03935. arXiv:2607.03935.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2303.17651. arXiv:2303.17651.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2411.00640. arXiv:2411.00640.

Xuying Ning, Dongqi Fu, Tianxin Wei, Hanqing Zeng, Yuanchen Bei, Bingxuan Li, Zihao Li, Qifan Wang, Xiang Shen, Yi-Fan Wu, Jia-Yi Liu, Hong Li, Yinglong Xia, Xiangjun Fan, Hanghang Tong, and Jingrui He. Evoharness-rl: Learning self-evolving runtime harness for long-horizon llm agents. arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2608.05446. arXiv:2608.05446.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2406.11695. arXiv:2406.11695.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2606.05922. arXiv:2606.05922.

Shishir Patil et al. Berkeley function-calling leaderboard: data and evaluator. https://github.com/ShishirPatil/gorilla, 2026. accessed 2026-09-12.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2305.15334. arXiv:2305.15334.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2305.03495. arXiv:2305.03495.

Yu Shang, Yu Li, Keyu Zhao, Li-Kai Ma, Jia-He Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2410.06153. arXiv:2410.06153.

Noah Shinn, Federico Cassano, Beck Labash, A. Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2303.11366. arXiv:2303.11366.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2305.16291. arXiv:2305.16291.

Z. Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2409.07429. arXiv:2409.07429.

Tianxin Wei, Zhan Shi, Minhua Lin, B. He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoît Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents. arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2608.15071. arXiv:2608.15071.

Yu-Hao Wu, Jingyuan Zhang, Jia-Jun Shi, Xin-Ping Lei, Qing-Shui Gu, Yuxuan Zhang, Zebing Wang, Chen He, Chenxi Huang, Maojia Song, Zhi-Yuan Zeng, Shao-Wen Wang, Jin-Kai Liu, Yunhan Shi, Jiaheng Liu, Shen Yan, Wen-Hao Huang, Ge Zhang, and Wenxin Zhang. Harnessdev: Can llms create and evolve their own agent harness? arXiv (Cornell University), 2026. doi: 10.48550/arxiv.2609.01437. arXiv:2609.01437.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. arXiv (Cornell University), 2023. doi: 10.48550/arxiv.2309.03409. arXiv:2309.03409.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Adriano Lieret, Shun-Yu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2405.15793. arXiv:2405.15793.

Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and W. Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. arXiv preprint arXiv:2410.04444, 2024. URL https://arxiv.org/abs/2410.04444. arXiv:2410.04444.

Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2406.07496. arXiv:2406.07496.

E. Zelikman, Eliana Lorch, L. Mackey, and A. Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. arXiv (Cornell University), 2023. doi: 10.48550/arXiv.2310.02304. URL https://arxiv.org/abs/2310.02304. arXiv:2310.02304.

Gui-Min Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. arXiv, 2025a. doi: 10.48550/arxiv.2502.04180. arXiv:2502.04180.

Jenny Zhang, Shengran Hu, Cong Lu, R. Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. arXiv (Cornell University), 2025b. doi: 10.48550/arxiv.2505.22954. arXiv: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. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2410.10762. arXiv:2410.10762.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, V. Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Han-Chen Li, Urmish Thakker, James Zou, and K. Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. arXiv (Cornell University), 2025c. doi: 10.48550/arxiv.2510.04618. arXiv:2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Gaetan Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In Proceedings of the AAAI Conference on Artificial Intelligence, 2024. doi: 10.1609/aaai.v38i17.29936.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. arXiv (Cornell University), 2022. doi: 10.48550/arxiv.2211.01910. arXiv:2211.01910.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. arXiv (Cornell University), 2024. doi: 10.48550/arxiv.2402.16823. arXiv:2402.16823.

A Pre-registration and Protocol Details

A.1 Pre-registration and the clean-stratum manifest

The document below registers the clean-pool evaluation before the new evaluations it specifies. The proposal corpus and existing A/B evaluations predate this registration and informed its design. It specifies the arms, strata, inclusion policy, tests, decision rule, and planned power. The original registration is reproduced below; the executed manifest distinguishes primary, expanded, separate abstention, and exploratory strata. The registration date and source document are included in the supplement.

Registered: 2026-09-12T01:17Z (UTC), before any new evaluation was run. Registered by: the experiment runner. This file is committed before results exist.

A.1.1 1. Hypothesis (single, directional)

`H1`: the population-level consensus harness (mechanism-level vote across ≥50% of independent evolution chains, inert components removed, parameters taken from the best-scoring chain) beats the bare scaffold (no instructions, one active component) on clean, never-evaluated held-out BFCL tasks, with the consensus arm costing more tokens.

Null hypothesis: the paired mean difference is 0.

A.1.2 2. Arms (fixed, no additions after seeing data)

arm	source	active components
`bare(no_instructions)`	the released rig	1
`pop:consensus`	the accompanying release (frozen; built from tag `bflm-*`, 7 chains, recurrence ≥0.50)	4
`checklist_accepted`	distilled from accepted edits, recurrence ≥0.25, top 8	2
`checklist_proposed`	distilled from proposed edits, recurrence ≥0.25, top 8	5

Task model: `Qwen3.8-27B`, temperature 0.3 (inherited from the harness spec), k = 3 samples/task.

A.1.3 3. Strata and exclusions --- declared BEFORE running

- Included (primary): BFCL strata that are (a) drawn from the `heldout` split (indices not in `range(N)[0::3]`), and (b) disjoint from every task used for evolution, gating, noise probing or dev evaluation. Overlap is checked programmatically against the id sets of all prior runs and recorded in the accompanying release.
- Excluded, by rule, from the primary number: any stratum taken from the `dev` split (e.g. prior subset C, offset 60, n=291). It is reported separately as an exploratory stratum and is never pooled into the primary estimate, because selecting it would amount to using the dev pool to manufacture significance.
- Excluded: `mgsm` (no tool calls; saturated for this task model, so it is used only as a no-regression probe), and prior subset B replication-1 (the released logs) is used only to estimate decision noise, not pooled twice; the two replications of subset B are averaged per task before pooling (see §5).
- No post-hoc dropping of tasks, categories or strata. If a whole stratum fails the client-error-rate guard (>20 % invalid calls) it is excluded and the failure is reported.

A.1.4 4. Primary endpoint and test

- Endpoint: per-task success rate averaged over k=3 samples (task success = all ground-truth calls matched by the BFCL AST subset matcher, verbatim from the official evaluator semantics).
- Primary estimate: paired mean difference (consensus − bare) pooled over all included clean strata, with a 95 % bootstrap CI (20 000 resamples over tasks, stratified by stratum) and an exact McNemar test on the binarised per-task outcomes.
- Decision rule, fixed now: significant $\Leftrightarrow$ $p < 0.05$ and the 95 % CI excludes 0. Both outcomes are reportable: if non-significant, the paper must say "direction consistent across all strata but did not reach the pre-declared threshold", and must report the realised MDE.

A.1.5 5. Statistical power --- declared BEFORE running

Discordance rate q measured on existing data (`pop:consensus` vs `bare`, per-task binary): A n=300 → q=0.0533; B n=102 → q=0.0392; C(dev) n=291 → q=0.0722. Planning value q = 0.060.

MDE = $(z_{0.975} + z_{0.80}) \cdot \sqrt{(q/n)} = 2.802 \cdot \sqrt{(q/n)}$:

pooled n	MDE (pp)
402 (clean A+B, status quo)	3.42
1 007 (goal target)	2.16
1 539	1.75
1 699 (all clean BFCL held-out)	1.66

Sample size needed for a given true effect at q=0.06: 754 tasks @ 2.5 pp, 1 178 @ 2.0 pp, 2 094 @ 1.5 pp --- i.e. the largest clean BFCL held-out pool available (~1 699) cannot resolve a 1.4 pp effect at 80 % power. This ceiling is itself a reportable result and must appear in the paper next to any non-significant outcome.

Replication-2 handling for subset B: the same 102 tasks were evaluated twice (`disjoint_subset` Δ=−0.59 pp, `disjoint3` Δ=+1.18 pp). The two replications are averaged per task before pooling so B contributes n=102, not 204. The 1.76 pp spread between two replications of the same comparison on the same tasks is reported in the paper as a direct measurement of the decision noise floor at n=102.

A.1.6 6. Secondary endpoints (declared, all reported regardless of sign)

1. Δ per stratum (sign consistency across strata --- the paper's existing 5/5 claim is extended, not replaced).
2. Tokens per task per arm (cost of the consensus arm); reported as a ratio, never as "free".
3. `checklist_accepted` and `checklist_proposed` vs bare on the same clean strata: does the "raw-proposal aggregation is harmful" negative result generalise to unseen categories?
4. Client-error rate per arm (batch-failure guard).

A.1.7 7. Planned strata (all held-out split, all verified disjoint in §3)

id	BFCL category	offset	n	new?
A	live_multiple	0	300	existing
B	live_multiple	300	102	existing (2 reps averaged)
S1	live_multiple	402	300	new
S2	live_simple	0	172	new (cross-category)
S3	parallel_multiple	0	133	new (cross-category)
S4	multiple	0	133	new (secondary, max power)
S5	simple_python	0	134	new (secondary, max power)
S6	parallel	0	133	new (secondary, max power)
S7	irrelevance	0	160	new (secondary, max power)
C	live_multiple (dev split)	60	291	existing, excluded from primary

Primary pooled n = 300 + 102 + 300 + 172 + 133 = 1 007. Max-power pooled n (all clean strata) = 1 007 + 133 + 134 + 133 + 160 = 1 669 [arithmetic note: the addends sum to 1 567; as executed, the abstention stratum S7 is pooled separately and the max-power pool is 1,407].

Stratum manifest as executed. Disjointness is verified programmatically before pooling: the pooling step exits non-zero on any task-id overlap between strata, rather than reporting a pooled estimate, which is why every pooled number here is accompanied by the manifest above.

stratum	category	split	offset	n	k	group
A	`bfcl_live_multiple`	heldout	0	300	3	primary
B	`bfcl_live_multiple`	heldout	300	102	5	primary
S1	`bfcl_live_multiple`	heldout	402	300	3	primary
S2	`bfcl_live_simple`	heldout	0	172	3	primary
S3	`bfcl_parallel_multiple`	heldout	0	133	3	primary
S4	`bfcl_multiple`	heldout	0	133	3	extend
S5	`bfcl_simple_python`	heldout	0	134	3	extend
S6	`bfcl_parallel`	heldout	0	133	3	extend
S7	`bfcl_irrelevance`	heldout	0	160	3	extend
C	`bfcl_live_multiple`	dev	60	291	3	exploratory

A.2 Corpus composition and the exclusion policy

Every corpus-level statistic in the main text (convergence, attractor, survival, dev/held-out deltas) is computed over chain sets selected by one machine-graded inclusion policy. The criteria are read from the EvoLog itself - no chain is named or excluded by hand anywhere in the pipeline, and a second, independent implementation re-derives the same grades inside the claim audit. Thresholds: flip-rate maximum 0.1, client-error guard 0.2.

Grades. L0 `hard_excluded`: zero baseline tokens, or no completed evolve round, or an invalid held-out batch (zero tokens / error rate above the guard), or a flip rate above twice the threshold. L1 `disclosed_retained`: flip rate above the threshold but at or below twice it - kept for fingerprint and decision statistics, excluded from delta means, disclosed in the main text. L2 `delta_clean`: flip rate within the threshold and a valid held-out batch - the only grade that may support a dev/held-out delta claim. Corpus-B (sensitivity): only L0's first two criteria apply, i.e. no noise-based exclusion at all.

chain	baseline tok	evolve rounds	flip rate (k=1)	held-out tok	client-err	grade
bflm-kimi-r0-no_instr-8873	87772	4	0.1167	0	0.1744	L0 hard-excluded
bflm-kimi-r0-swe_agen-8873	103177	4	0.1	219010	0.0388	L2 delta-clean
bflm-kimi-r1-no_instr-8873	86175	4	0.1	221296	0.0711	L2 delta-clean
bflm-kimi-r1-swe_agen-8873	103897	2	0.15	None	-	L1 disclosed-retained
bflm-qwen-r0-no_instr-8873	87445	2	0.1333	None	-	L1 disclosed-retained
bflm-qwen-r0-swe_agen-8873	67800	3	0.25	None	-	L0 hard-excluded
bflm-qwen-r1-swe_agen-8873	0	1	0.0	None	-	L0 hard-excluded

Selected sets for `bflm-*`: corpus-A (decision/fingerprint) n=4, corpus-B (sensitivity) n=6, L2 (delta means) n=2; hard-excluded: `bflm-kimi-r0-no_instr-8873`, `bflm-qwen-r0-swe_agen-8873`, `bflm-qwen-r1-swe_agen-8873`.

Sensitivity to the exclusion policy (bflm-*). Corpus-A is the paper's primary scope; corpus-B drops no chain for noise reasons. Every quantity keeps its sign and its order.

quantity	corpus-A (n=4)	corpus-B (n=6)			
coarse Jaccard, proposed	0.4414	0.3613			
coarse Jaccard, accepted	0.1729	0.1781			
fine Jaccard, proposed	0.0876	0.0866			
fine Jaccard, accepted	0.0	0.0205			
within-seed final-mechanism similarity	0.6875	0.5958			
across-seed final-mechanism similarity	0.1979	0.2463			
attractor strength (	shared	/	union	)	0.111	0.111
survival, component level	0.8571	0.9375			
survival, all keys (diluted)	0.5455	0.625			

Chains in corpus-A:. Of these 2 are L1 disclosed-retained (flip rate [0.15, 0.1333]), and 2 ran the full protocol with a recorded held-out batch.

Consensus artifact is invariant to the policy. Rebuilding the published artifact from corpus-A instead of from all chains that ran on this category gives a behaviourally identical specification: same mechanisms (action_format@context_post, decompose@task_start, system_prompt@task_start, tool_doc_augment@tool_pre), same parameters. The comparison uses the behavioural projection (kind/hook/params) because the frozen interpreter dispatches every processor by kind and never by list position, so emission order is inert. corpus-A rebuilt == corpus-B rebuilt: True; published artifact == corpus-A rebuild: True.

Delta-grade statistics (L2 only). Over the 2 chains whose flip rate is within the threshold and whose held-out batch is valid: mean dev delta +0.0667 (range [0.0583, 0.075]), mean held-out delta +0.0416 (range [0.0083, 0.075]). Chains: `bflm-kimi-r0-swe_agen-8873`, `bflm-kimi-r1-no_instr-8873`.

These are the chains' own in-loop spot-checks (dev n=60 at k=1, held-out n=120 at k=1), so they resolve only about ±12.7 pp and ±9.0 pp respectively. The decisive re-evaluation of the same artifacts at n=300, k=3 measured a loss; the in-loop checks could not have seen it. That gap is the paper's central methodological point.

chain	baseline tok	evolve rounds	flip rate (k=1)	held-out tok	client-err	grade
p1c-kimi-r0-5365	426552	3	0.0	711603	0.0105	L2 delta-clean
p1c-kimi-r1-5365	375154	3	0.0	None	-	corpus-A
p1c-kimi-r2-5365	401323	3	0.0	583624	0.0047	L2 delta-clean
p1c-kimi-r3-5365	379204	3	0.0	496145	0.0102	L2 delta-clean
p1c-qwen-r0-5365	375976	0	0.0	None	-	L0 hard-excluded
p1c-qwen-r1-5365	282269	0	0.125	None	-	L0 hard-excluded
p1c-qwen-r2-5365	314698	0	0.0625	None	-	L0 hard-excluded

Selected sets for `p1c-*`: corpus-A (decision/fingerprint) n=4, corpus-B (sensitivity) n=4, L2 (delta means) n=3; hard-excluded: `p1c-qwen-r0-5365`, `p1c-qwen-r1-5365`, `p1c-qwen-r2-5365`.

Sensitivity to the exclusion policy (p1c-*). Corpus-A is the paper's primary scope; corpus-B drops no chain for noise reasons. Every quantity keeps its sign and its order.

quantity	corpus-A (n=4)	corpus-B (n=4)			
coarse Jaccard, proposed	0.4668	0.4668			
coarse Jaccard, accepted	0.12	0.12			
fine Jaccard, proposed	0.0502	0.0502			
fine Jaccard, accepted	0.0	0.0			
within-seed final-mechanism similarity	0.3254	0.3254			
across-seed final-mechanism similarity	n/a	n/a			
attractor strength (	shared	/	union)	0.125	0.125
survival, component level	1.0	1.0			
survival, all keys (diluted)	0.8571	0.8571			

Chains in corpus-A:. Of these 0 are L1 disclosed-retained (flip rate none), and 3 ran the full protocol with a recorded held-out batch.

Delta-grade statistics (L2 only). Over the 3 chains whose flip rate is within the threshold and whose held-out batch is valid: mean dev delta +0.0260 (range [0.0, 0.0469]), mean held-out delta -0.0486 (range [-0.0625, -0.0208]). Chains: `p1c-kimi-r0-5365`, `p1c-kimi-r2-5365`, `p1c-kimi-r3-5365`.

These are the chains' own in-loop spot-checks (dev n=60 at k=1, held-out n=120 at k=1), so they resolve only about ±12.7 pp and ±9.0 pp respectively. The decisive re-evaluation of the same artifacts at n=300, k=3 measured a loss; the in-loop checks could not have seen it. That gap is the paper's central methodological point.

A.3 Evo-Rig, environments, gates and gates' statistics

A.3.1 Bounded, typed edit surface

A harness is a JSON document. It is an ordered list of processors attached to one of 8 lifecycle hooks (`task_start`, `context_pre`, `context_post`, `step_pre`, `step_post`, `tool_pre`, `task_end`, `control`), plus scalar control (`max_steps`) and sampling parameters. Each processor `kind` declares its legal hooks and a parameter schema. Thirteen kinds are implemented (system_prompt, decompose, history_limit, obs_format, action_format, error_reframe, tool_doc_augment, retry_policy, self_check, early_stop, memory_cheatsheet, output_format, answer_guard). Six edit operators act on them (`add`, `delete`, `modify`, `reorder`, `toggle`, `param`). Every proposed edit passes `validate` against the type system before it can be applied. Out-of-contract edits (illegal hook, unknown parameter, out-of-range enum) are rejected and logged with the reason.

Why bounded rather than free-form code self-modification. Three properties we need are unavailable otherwise. (i) Exact reconstruction: any historical H_t is re-derivable by replaying the logged edit sequence (we verify this mechanically; the check ships with the supplement). (ii) Static gating: leakage and contract violations are detectable without executing anything. (iii) Attribution: a diff is a set of typed component changes, so component-wise ablation is well defined. The cost is expressivity: edits that require novel control flow cannot be expressed. We treat this as the correct trade for a measurement study and note it as a limitation (Limitations).

A.3.2 Frozen runtime, evolvable spec

The interpreter (the released rig) is frozen. It implements the loop: render observation → call M with the tool schema → parse tool calls (with optional repair) → execute → reframe errors → repeat until the model stops calling tools, `max_steps` is hit, or an early-stop trigger fires; then optionally run self-check rounds; then verify. Because the interpreter never changes, all behavioural difference between harness versions is attributable to the spec, and because error-code semantics are not embedded in the interpreter, a chain can only exploit them by writing decode rules into `error_reframe.params.rules` --- the fix must be invented, not looked up. This is the single most important design decision for validity: had we hard-coded hint tables in the runtime, "evolution" would have been measuring our own engineering.

A.3.3 Evolver, Critic, and four gates

The meta-model receives (a) the current spec JSON and its brief, (b) a compressed evidence digest, (c) the archive of previously proposed edits with their realized deltas, and (d) the type system documentation, and must return edits with `rationale` and `expected_effect` per edit. Digests come in six evidence grades E0--E5 (outcome scalars → full trajectories → divergence-point slices → minimal counterfactuals), which doubles as the ablation knob for "how much trace does an evolver need". An optional Critic reviews the manifest against the evidence for over-generalization, hack-shaped edits, and instance leakage, with at most one revision round.

Four gates decide acceptance, and LLM judgment is never part of the decision:

- G1 manifest: type validity + presence of rationale/expected_effect.
- G2 literal density (hard): the edit text must not contain any dev-instance literal (IDs, amounts, account numbers) extracted from the dev split. This blocks the cheapest form of overfitting and a common reward-hacking channel.
- G3 see-saw: no dev task solved under H_t may become unsolved under H_{t+1}, with tolerance derived from the measured noise floor rather than hardcoded.
- Smoke: 3 tasks × 1 sample must execute without crash, client error, or contract violation --- this runs before the full dev evaluation and exists because evolvers invent placeholder names (e.g. `{error}` where the contract is `{err}`) that would otherwise crash a full evaluation round.

A.3.4 Adaptive tie-break resampling

With n dev tasks and k samples, the score resolution is $1/n$ and the noise floor is comparable, so the naive rule "accept iff $\Delta > 0$" both rejects real gains and accepts noise. When $|\Delta| \leq 1.5/n$, Evo-Rig draws `tie_extra_k` additional samples for both the candidate and the incumbent and re-decides on the pooled means. Every tie-break event (first-pass delta, band, pooled delta, extra tokens) is logged, so the reader can audit how often the decision was noise-limited. Across the corpus, 17 of 40 decisions (42.5%) required tie-break, consuming 12.04M extra tokens (≈708K per intervention). In at least one logged case the resampling reversed the verdict: first-pass $\Delta = +0.016$ became pooled $\Delta = -0.008$ and the edit was rejected; another candidate's first-pass $\Delta = +0.094$ settled at $+0.078$. Under the naive rule the first edit would have entered the archive as an improvement.

A.3.5 EvoLog

Append-only JSONL with record types `run / round / edit / rollout / gate / cost`, plus the full spec JSON of every version. Rollouts store step-level traces (model text, tool calls, observations with error flags, tokens, latency, contract violations). This is the paper's reproducibility artefact and, we argue, a community resource: public corpora of self-evolution decisions --- including rejected edits and why --- do not exist.

A.4 ToolOrbit and the friction layer

Environment. Three business domains (banking transfer, order fulfilment, flight rebooking), 90 programmatically generated tasks (30 dev / 60 held-out, deterministic by seed), tiered by subgoal count (1--3), tool-set size (3--6 including distractors), and dual-control requirements (one tier hides a required parameter that only the simulated customer knows, retrievable via `ask_user`). Scoring is pure state assertion: expect conditions on the final database, forbid conditions (e.g. the audit table must not become non-empty), table-level no-write invariants against the initial snapshot, and must-not-call constraints on high-risk tools (`set_balance`, `delete_order`, `wipe_audit`, `override_fare`). The last three exist so that shortcuts which satisfy the positive assertions still fail.

Validity checks. A friction-aware oracle solver achieves 90/90 on both splits (before and after friction is enabled), which certifies solvability and doubles as the Oracle Plan upper bound. A bare seed harness scores 0.526 ± 0.033 (6 repetitions, dev n=16×k=2) against an oracle of 1.00, i.e. ~47 pp of calibrated headroom.

Friction layer. A wrapper injects six failure modes without touching the environment's logic (the "wrap, don't rebuild" pattern). Each mode is curable by a specific processor kind, which is what makes "which edit fixed what" answerable:

Friction	Mechanism	Cured by	Observed symptom in traces
transient	18% of calls fail with `E-9001` and no side effect	`retry_policy`	write silently skipped → final quantity mismatch
riskgate	missing prerequisite query → cryptic code (`E-4031`, `E-6202`, …)	`error_reframe.rules` (semantics must be inferred by the evolver)	agent loops the same call until `max_steps`
silentnoop	prerequisite unmet → tool returns fake `OK`, state unchanged	`self_check`, `answer_guard`	"success" followed by assertion failure
verbose	read tools append 22 lines of noise	`history_limit`, `obs_format`	context and token inflation
dualcontrol	required parameter only obtainable via `ask_user`	`tool_doc_augment`, `decompose`	fabricated value → `contains_field` failure
distractor	high-risk shortcut tools that write the audit table	(negative assertions)	measures reward-hacking propensity

A.5 Verifiers, answer channels and saturation checks

A.5.1 Verifier extensions (BFCL AST subset matching)

The official BFCL matcher compares each predicted call against a list of acceptable calls, requiring the predicted arguments to be a subset of the ground truth. Three cases in the public data break a naive port; each is pinned by a unit case that ships with the supplement:

1. Nested containers. Ground truth may contain `dict`/`list` parameters; comparison recurses and coerces `"123"` vs `123` and `"true"` vs `True`.
2. Optional parameters. Omitting an optional parameter is correct behaviour; requiring it penalises the very precision the benchmark rewards.
3. Parallel categories. `parallel` and `parallel_multiple` require several calls in one turn, so the episode terminates after the full turn rather than after the first call.

A.5.2 MGSM: answer channels and saturation

MGSM-en is scored by exact numeric match on an extracted final answer, and the scorer reads both channels through which the policy can answer: the structured `submit_answer` tool call and the free-text answer the task prompt also permits. The benchmark is saturated for this task model: the bare scaffold scores 97.2% over all 250 items (per-25-item accuracy 0.92-1.00) and 96.4% on the 84-item dev stride, where every arm --- bare and the four transcribed scaffolds --- sits inside 96.4-98.8% and no paired McNemar comparison against bare reaches significance (all $p \geq 0.5$). It therefore cannot show a harness gain; it can only show harm. The paper uses it strictly as a no-regression probe, which is what the measurement below is.

Protocol rule for every benchmark integration in this paper. (i) an answer-channel audit --- does the verifier read every channel the policy can use?; (ii) a saturation probe over the entire pool, bare scaffold and best transcribed scaffold; (iii) a noise-floor probe before any hypothesis test. A number computed on an integration that fails any of the three is not reported.

A.5.3 The no-regression probe: cross-domain transfer to MGSM

Since this benchmark is saturated (above), it cannot show a gain but it can show harm, which makes it the right instrument for one question: does a harness evolved for tool-calling damage unrelated reasoning? We evaluate every chain's final spec on the MGSM held-out stride without any adaptation, against the native references measured inside the same evaluation (Table 11), so the comparison is paired on one task list with one verifier.

Table 11 --- Cross-domain transfer, BFCL `live_multiple` → MGSM-en (final harness of each valid chain, no adaptation; n=60, k=1; all arms measured in the same run so the comparison is paired).

arm	fitness	95% CI	tok/task
native bare scaffold on MGSM	95.0%	[88.3, 100.0]	708
native `swe_agent_aci` on MGSM	98.3%	[95.0, 100.0]	872
native `codex` on MGSM	93.3%	[86.7, 98.3]	712
transferred evolved harnesses (6 chains, mean)	91.7%	[88.1, 95.0]	848 (1.20× the bare scaffold)
Δ transferred vs native-bare	−3.3 pp	---	+20% tokens

Reading: transplantation is mildly harmful and more expensive, but not catastrophic --- no arm collapses, the worst single transferred harness is 85.0% (CI [75.0, 93.3]) against a native bare 95.0%. So a harness evolved for tool-calling degrades unrelated reasoning by a few points and costs 20% more tokens. That is the failure mode practitioners actually see: not a crash, but a silent few-point tax on everything else in the system. It also bounds what the checklist deliverable can promise --- shipping an evolved harness across domains requires the no-regression probe, not faith.

Why we report a saturated benchmark at all. Because an evolved harness that is never tested outside its domain is an unaudited artifact. A single saturated probe costs ~100 episodes (≈0.1M tokens) and rules out the most damaging failure mode --- silent degradation on unrelated work --- that accuracy-only, single-domain evaluation cannot detect.

B Models and Compute

B.1 Models, roles and access

role	checkpoint (public name)	used for	measured JSON reliability	latency (s)	accessed
`task`	Qwen3.8-27B	agent rollouts in every evaluation (frozen base model)	clean	1.4	2026-09-11
`task_strong`	Qwen3.8-Max	capability-ladder arm (§6.3)	clean	1.5	2026-09-11
`meta_a`	Kimi-K3	evolver / proposer in the main corpus	clean	5.2	2026-09-11
`meta_b`	Qwen3.5-397B-A17B	evolver, second family (cross-family ablation)	unsupported	2.2	2026-09-11
`critic`	Kimi-K2.7-Code	critic inside the propose→critique→revise loop	clean	2.1	2026-09-11
`judge`	Kimi-K3	adjudication (used only for diagnostics, never in acceptance)	clean	5.2	2026-09-11
`labeler_1`	Kimi-K2.7-Code	edit-semantics labeler 1	clean	2.1	2026-09-11
`labeler_2`	Qwen3.5-397B-A17B	edit-semantics labeler 2 (double labelling for κ)	unsupported	2.2	2026-09-11

Same checkpoint, more than one role: `Kimi-K3` serves `meta_a`, `judge`; `Qwen3.5-397B-A17B` serves `meta_b`, `labeler_2`; `Kimi-K2.7-Code` serves `critic`, `labeler_1`. Role separation: task / meta / critic are assigned to different model families so that the loop cannot grade its own homework; this is the standard defence against the self-distillation objection, and it is a design constraint, not a result. Capability screen: every endpoint was screened on four probes (plain completion, tool call, strict JSON, 15k-token context) before being assigned a role (0 models, all reachable); one endpoint was disqualified from every structured role because it emits chain-of-thought into the answer channel and never closes it, which silently corrupts parsing. That screen is part of the protocol because an unstable meta-model produces an unstable archive, and the failure mode is invisible in the score. Access: models are served through an internal inference gateway; no weights were trained, fine-tuned, or modified. Sampling parameters are recorded per harness spec (temperature 0.3 in every spec we evaluate). The gateway identifier suffix used in our internal configuration has been removed throughout this paper so that internal service naming does not identify the authors.

Checkpoint names in this paper are the public ones. The supplement's model configuration additionally records, for each endpoint, the internal service label (public name plus a deployment-suffix that denotes a serving route, not a different checkpoint), the role assignment (task / meta / critic) and the verified capability flags used to screen it; the mapping is one-to-one, so every number here can be reproduced against the public checkpoints named above (8 endpoints).

B.2 Compute and cost

No weights were trained or fine-tuned anywhere in this study; every experiment is a sequence of API calls to the served checkpoints listed above, and every comparison freezes one task model. Measured spend for the archived campaigns, probes and replications preceding the supplementary follow-up: 110.4 million tokens (approximately 18.03 USD list-price equivalent), 5.3 wall-clock hours, zero GPU-hours. Per-task token costs of each arm are tabulated with the results themselves (Appendix C.1); the evolution-side token inflation of single chains is analysed in §6.2. The supplementary follow-up reports its additional valid-episode token count separately.

The decisive evaluation is the largest single line item: 7 200 episodes over 8 arms (n=300, k=3), and the consensus artifact is then scored once more on every clean stratum (1,407 paired tasks in the max-power pool).

C Additional Experimental Results

C.1 Full per-stratum results

Every arm of every clean stratum, as evaluated. Delta is paired against the bare scaffold within the stratum, CI is a stratified bootstrap over tasks, p is exact McNemar, and `cer` is the client-error rate (the guard that aborts an evaluation batch above 20 percent rather than scoring it). Strata A and B are the two campaigns of the decisive experiment and carry their archived names in the per-block headings.

C.1.1 Stratum A --- bfcl live multiple (n=300, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
`pop:consensus`	82.00%	+1.78 pp	[-0.44, +4.11]	0.455	1749	0.0533
`checklist_proposed`	74.00%	-6.22 pp	[-9.00, -3.67]	1e-05	1657	0.0867
`checklist_accepted`	80.11%	-0.11 pp	[-1.78, +1.56]	0.754	1427	0.0333
`pop:consensus`	82.00%	+1.78 pp	[-0.44, +4.11]	0.455	1749	0.0533

C.1.2 Stratum B --- bfcl live multiple (n=102, k=5, split heldout, offset 300)

arm	accuracy	delta	95% CI	p	tok/task	q
`pop:consensus`	81.18%	+0.29 pp	[-1.96, +2.55]	1	1859	0.0294
`checklist_proposed`	78.63%	-2.25 pp	[-4.71, +0.10]	1	1773	0.0294
`checklist_accepted`	80.49%	-0.39 pp	[-1.27, +0.39]	1	1579	0.0098
`pop:consensus`	81.18%	+0.29 pp	[-1.96, +2.55]	1	1859	0.0294

C.1.3 Stratum S1 --- bfcl live multiple (n=300, k=3, split heldout, offset 402)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	77.00%	+0.89 pp	[-0.44, +2.33]	0.125	1811	0.0233
checklist_proposed	73.89%	-2.22 pp	[-4.22, -0.22]	0.424	1714	0.0467
checklist_accepted	75.33%	-0.78 pp	[-2.33, +0.67]	1	1498	0.0233
pop:consensus	77.00%	+0.89 pp	[-0.44, +2.33]	0.125	1811	0.0233

C.1.4 Stratum S2 --- bfcl live simple (n=172, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	88.37%	+3.49 pp	[+0.58, +6.59]	0.0117	930	0.0640
checklist_proposed	82.95%	-1.94 pp	[-5.04, +0.97]	1	911	0.0640
checklist_accepted	86.24%	+1.36 pp	[-0.58, +3.49]	0.125	721	0.0407
pop:consensus	88.37%	+3.49 pp	[+0.58, +6.59]	0.0117	930	0.0640

C.1.5 Stratum S3 --- bfcl parallel multiple (n=133, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	79.70%	+1.75 pp	[-1.75, +5.76]	0.774	1368	0.0902
checklist_proposed	68.17%	-9.77 pp	[-15.04, -4.51]	0.0347	1367	0.1729
checklist_accepted	78.70%	+0.75 pp	[-1.50, +3.26]	1	1093	0.0376
pop:consensus	79.70%	+1.75 pp	[-1.75, +5.76]	0.774	1368	0.0902

C.1.6 Stratum S4 --- bfcl multiple (n=133, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	92.98%	+1.25 pp	[-1.00, +3.51]	0.625	1097	0.0301
pop:consensus	92.98%	+1.25 pp	[-1.00, +3.51]	0.625	1097	0.0301

C.1.7 Stratum S5 --- bfcl simple python (n=134, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	92.54%	+1.24 pp	[-1.74, +4.48]	0.508	820	0.0672
pop:consensus	92.54%	+1.24 pp	[-1.74, +4.48]	0.508	820	0.0672

C.1.8 Stratum S6 --- bfcl parallel (n=133, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	93.23%	+2.76 pp	[+0.50, +5.51]	1	1006	0.0376
pop:consensus	93.23%	+2.76 pp	[+0.50, +5.51]	1	1006	0.0376

C.1.9 Stratum S7 --- bfcl irrelevance (n=160, k=3, split heldout, offset 0)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	80.42%	+5.62 pp	[+2.29, +9.17]	0.00418	1305	0.1000

C.1.10 Stratum C --- bfcl live multiple (n=291, k=3, split dev, offset 60)

arm	accuracy	delta	95% CI	p	tok/task	q
pop:consensus	80.41%	+4.12 pp	[+1.95, +6.41]	0.00149	1758	0.0722
checklist_proposed	72.62%	-3.67 pp	[-6.53, -1.03]	0.0433	1651	0.0859
checklist_accepted	77.55%	+1.26 pp	[-0.57, +3.09]	0.302	1458	0.0515

Exploratory. Drawn from the `dev` stratum (offset 60): no chain ever evolved on it, but it shares the dev pool, so the pre-registration excludes it from every headline number.

C.1.11 The decisive 8-arm experiment, in full

arm	fitness	delta	95% CI	p	tok/task	components
bare(no_instructions)	80.33%	+0.00 pp	[+0.00, +0.00]	1	1431	1
checklist_accepted	79.44%	-0.89 pp	[-2.44, +0.67]	0.5078	1430	2
pop:proposed_inertf	78.78%	-1.56 pp	[-3.89, +0.67]	0.3593	1536	2
pop:proposed	78.56%	-1.78 pp	[-4.11, +0.44]	0.3593	1537	4
pop:gated	77.00%	-3.33 pp	[-5.67, -1.11]	0.04904	1534	2
evolved:-no_instr-8873	75.78%	-4.56 pp	[-7.00, -2.22]	0.00154	1577	4
evolved:-swe_agen-8873	75.67%	-4.67 pp	[-6.89, -2.56]	0.00739	1590	8
checklist_proposed	73.44%	-6.89 pp	[-9.78, -4.22]	3e-05	1659	5

Residual noise from behaviourally-equivalent arms. 3 arms that a single-step environment cannot distinguish score 77.00 / 78.56 / 78.78%, i.e. SD 0.79 pp and range 1.78 pp. This is why we report an MDE next to every delta in this paper.

C.2 Prose compilation and registered comparisons

Union and recurrent prose are implementations of cross-chain proposal compilation. They were originally evaluated as representation controls; presenting them as methods is a retrospective interpretation of those experiments. The registered contrasts and evaluated artifacts are unchanged. Matched prose remains a length control.

Construction (deterministic, from the EvoLog). Union prose scans all archived edit logs, including BFCL, ToolOrbit, MGSM and supplementary runs. This scope is broader than the BFCL population used for typed consensus. It extracts string-valued fields named `text`, `rules`, `prompt`, `hint`, or `examples` from proposals, whether accepted or rejected. It strips whitespace and drops blocks shorter than 40 characters. A whitespace-normalized 200-character prefix is the deduplication key; the first encountered block is retained. Blocks are then sorted lexically. The compiler takes the leading blocks until the next block would exceed a 6,000-character content cap, then stops. It joins them with bullet prefixes and blank lines into one `system_prompt` component at `task_start`. There is no model-based rewriting or conflict resolution.

Recurrent prose. The same extractor and deduplicator operate on the selected typed artifact's processor parameters. Thus recurrent prose inherits typed consensus's source population, recurrence and coverage rules, and development-ranked implementation choices. It is not a union of all text ever proposed for recurrent mechanism kinds. The evaluated artifact contains 1 text block; the representation no longer preserves the original hooks.

Matched-prose control. A seeded permutation of union blocks supplies a subset. Its character budget uses a fraction of archived union content, calibrated from incremental inference-token costs on stratum A. The compiler skips oversized blocks after selecting the first block. This is approximate budget matching, not equal realized inference tokens.

variant	source	prompt chars	active components	role
Union prose	broader proposal archive	6031	1	prose compilation
Recurrent prose	selected typed artifact	716	1	prose compilation
Matched prose	seeded subset of union	1389	1	approximate length control

Prompt lengths include separators and bullet prefixes. The offline reconstruction audit reproduces all evaluated prompt strings exactly and records a hash and source-chain list for each block in the release. These provenance differences preclude treating union versus typed consensus as a representation-only ablation.

All three were evaluated on the pre-registered clean primary pool (1007 tasks, 5 strata, k as in the stratum, frozen task model), paired against the archived bare scaffold's per-task scores on the same task ids. Pre-registration and decision rules: Appendix A.1 (registered before these paired prose evaluations, after earlier union-prose measurements); the binary-endpoint planning MDE at this n is 1.95 pp.

stratum	n	k	arm	accuracy	Δ vs bare	95% CI	McNemar p	win/lose	tok/task
A	300	3	`prose:union`	82.78%	+2.56 pp	[+0.11, +5.00]	0.21	15/8	2,747
A	300	3	`prose:consensus`	82.33%	+2.11 pp	[-0.11, +4.33]	0.4807	11/7	1,610
A	300	3	`prose:matched`	81.33%	+1.11 pp	[-1.33, +3.56]	1	12/11	1,832
B	102	5	`prose:union`	80.59%	-0.59 pp	[-3.33, +2.16]	0.625	3/1	2,898
B	102	5	`prose:consensus`	81.18%	-0.00 pp	[-2.94, +2.75]	0.4531	5/2	1,753
B	102	5	`prose:matched`	80.20%	-0.98 pp	[-4.12, +1.96]	1	3/3	1,954
S1	300	3	`prose:union`	77.11%	+1.00 pp	[-0.89, +3.00]	0.3018	10/5	2,854
S1	300	3	`prose:consensus`	77.33%	+1.22 pp	[-0.33, +2.78]	0.3877	8/4	1,717
S1	300	3	`prose:matched`	76.00%	-0.11 pp	[-2.22, +1.89]	0.6072	9/6	1,930
S2	172	3	`prose:union`	88.37%	+3.49 pp	[+0.39, +6.78]	0.01294	12/2	2,014
S2	172	3	`prose:consensus`	86.63%	+1.74 pp	[-1.16, +4.65]	0.3877	8/4	890
S2	172	3	`prose:matched`	85.85%	+0.97 pp	[-2.13, +4.26]	0.4545	10/6	1,101
S3	133	3	`prose:union`	83.21%	+5.26 pp	[+1.50, +9.27]	0.01172	10/1	2,430
S3	133	3	`prose:consensus`	83.46%	+5.51 pp	[+1.50, +10.03]	0.03857	10/2	1,281
S3	133	3	`prose:matched`	80.70%	+2.76 pp	[-1.75, +7.27]	0.2266	8/3	1,508

Pooled over the clean primary pool:

These archived prose intervals bootstrap paired task differences over the pooled task list. Unlike the typed clean-pool analysis, they do not hold stratum sizes fixed in each bootstrap draw.

arm	n	Δ vs bare	95% CI	positive/negative task deltas
`prose:union`	1007	+2.29 pp	[+1.06, +3.54]	89/59
`prose:consensus`	1007	+2.02 pp	[+0.86, +3.17]	83/46
`prose:matched`	1007	+0.73 pp	[-0.56, +2.03]	81/78

Pre-registered verdicts (applied verbatim).

- Primary contrast, `prose:consensus` minus `prose:union`: -0.27 pp [-1.19, +0.62], p0.143, 14/24 wins-losses. Inside the ±1.95 pp band, therefore not separable at this n: filtering may not be claimed as the causal ingredient of the prose-level gain.
- Secondary contrast, `prose:matched` minus `prose:union`: -1.56 pp [-2.47, -0.66], p0.00289. Verdict per the rule: length is not the driver (the budget-matched arm retains the gain); point estimate -1.56 pp, CI [-2.47, -0.66], p=0.00289
- Exploratory, not pre-registered (`prose:consensus` minus `prose:matched`): +1.29 pp [+0.32, +2.27], p0.184. We report it because it is suggestive - at a comparable token budget, choosing content beat truncating it - and we label it exploratory because no decision rule was fixed for it in advance.

Repeated measurement. The identical `prose:union` specification on the identical stratum-A task list was evaluated 3 times in 3 independent processes: [81.00, 82.67, 82.78]. The observed range is 1.78 pp (SD 0.997) at n=300, k=3. This documents variability of a fixed artifact. Failure to separate the prose variants under the registered contrast does not establish equivalence.

Interpretation: proposal reuse, selection, and cost.

1. Prose compilation recovers useful proposal content. Union prose improves sampled accuracy by +2.29 pp [+1.06, +3.54]. Recurrent prose also has a positive pooled effect. These are paired comparisons against the archived baseline; they do not establish that prose outperforms typed consensus, whose estimate is +1.65 pp.
2. The registered contrast does not isolate a recurrence advantage. Recurrent and union prose are not separated by the registered decision rule. This is not an equivalence test. Their input scopes and selected content differ, and the typed comparisons additionally change execution structure. No single construction choice is identified as the cause of the gains or of the checklist baseline's degradation. Both prose and typed artifacts can be reconstructed; typed components additionally support processor-level static validation and coverage analysis.
3. Inference cost differs. On stratum A, union prose uses 2,747 tokens per task, recurrent prose 1,610, and the archived bare baseline 1,431. These are inference costs only. Compilation requires no extra model calls, but the input chains have a generation cost. These archived prose evaluations cover the primary pool; a separate frozen-artifact expansion follow-up is reported next.

C.3 Contemporaneous expansion follow-up

This supplementary experiment was specified before its new API evaluations, after the original primary results were known. It is not part of the original pre-registration. The task list comprises the existing clean S4, S5, and S6 strata. All five arms are evaluated afresh on the same tasks and sampling protocol; archived baseline scores are not reused.

The released protocol freezes the exact artifacts, task IDs, model, samples, and analysis. Union, recurrent, and matched prose use the already evaluated instruction text; typed consensus uses the released artifact. No artifact is tuned on the follow-up. All categories and arms are reported without outcome-based stopping.

There are 400 tasks with 3 samples per arm, yielding 6,000 valid episodes. Caching is disabled. Deterministically shuffled arm batches share one client rate limiter. Batch concurrency was increased to hide service latency without changing decoding or samples. A batch containing client errors is invalidated and rerun; incomplete batches are resumed from checkpoints. The retained records have no client errors. These exclusions concern transport validity, not task success.

Mean sampled accuracy is averaged over tasks. Pooled confidence intervals use 20,000 paired bootstrap draws, resampling within each stratum and weighting by task count. Exact McNemar tests use the separate thresholded endpoint. Per-category estimates are descriptive and multiple arm comparisons are not multiplicity-adjusted.

Harness	Accuracy (%)	Paired delta (pp)	95% CI	McNemar p	Tokens / bare
Bare	91.92	+0.00	[+0.00, +0.00]	1	1.00
Union prose	92.42	+0.50	[-0.92, +2.00]	0.4545	2.75
Recurrent prose	92.58	+0.67	[-0.58, +2.00]	0.581	1.23
Typed consensus	92.33	+0.42	[-1.00, +1.83]	0.7905	1.32
Matched prose	90.67	-1.25	[-2.92, +0.42]	0.8238	1.52

Completed valid episodes used 6,921,805 tokens in total. This count excludes transport retries and interrupted batches, whose complete overhead was not retained. It is an additional evaluation cost, not part of proposal generation. The original archived pool estimates are unchanged.

The follow-up evaluates category transfer, not isolated representation or filter effects. The source-scope differences between compilation variants remain. It is not an independent task-model replication.

C.4 Companion results: budget allocation, capability ladder and transfer

C.4.1 H4: how to spend a budget of k rollouts

One harness sampled k times vs a portfolio of k different harnesses, equal token budget (5771 tokens at k=4), n=60 tasks, 2 samples per (harness, task).

arm	meaning	accuracy at k=4
`A_same_harness_k_samples`	k samples of ONE harness (repeated sampling)	0.600
`B_portfolio_k_harnesses`	k DIFFERENT harnesses (portfolio)	0.750
`C_hybrid`	hybrid allocation	0.783
`D_random_route`	random routing to a harness	0.533
`D_oracle_route`	oracle routing (upper bound)	0.750
`E_oracle_best_of_k`	oracle best-of-k	0.600

McNemar (portfolio vs repeated sampling): p=0.022; difference = 0.1500.

Scope of this arm. H4 is supported only by the k=4 rows above, where every arm draws the same number of samples per (harness, task); a comparison in which the arms draw unequal sample counts cannot separate budget allocation from sampling depth, and is not used here.

C.4.2 H5/H6: error correlation and disagreement as a signal

- error correlation within one harness $\rho = 0.7762$ vs across harnesses $\rho = 0.5271$ ($\Delta = +0.2491$, SDI = 0.4729) -> H5 PASS
- AUROC(harness disagreement predicts failure) = 0.4629 vs AUROC(seed variance) = 0.5029, gain = -0.04 -> H6 FAIL: diversity does not buy abstention

C.4.3 Capability ladder (H3)

Three frozen task models, same environment, same gates, same seed scaffold, T=3, held-out n=80, k=1 (MDE $\approx \pm 15.7$ pp at q=0.25, so the held-out rows are individually unresolvable and only the large dev gain is above the floor).

frozen task model	dev baseline	dev delta	held-out delta
`Qwen3.5-122B-A10B` (mid)	78.1%	+1.0 pp	-5.0 pp
`Qwen3.8-Max` (least headroom)	65.6%	+19.8 pp	+5.0 pp

The prediction was that scaffold value falls as base-model capability rises. On this category the measured baselines are not even ordered by parameter count, and the arm with the LOWEST baseline gained the most: gain tracks headroom on the target distribution, not model scale.

C.4.4 Transfer and the published-harness control

measurement	value	source
harness evolved on `live_multiple`, evaluated on MGSM-en	-3.89 pp vs the target domain's own bare scaffold, tokens 1.20 x	the accompanying release
`bfcl_live_multiple` (five transcribed seed harnesses, one category)	74.0% (bare) / 79.0% (SWE-agent ACI) / 80.0% (Codex CLI) / 75.0% (Aider) / 75.0% (vendor tool-format); n=100; smallest p vs bare: `Codex CLI` (p=0.18, higher than bare); tokens +6% to +33%	the accompanying release
`bfcl_irrelevance` (abstention category)	73.8% (bare) / 75.0% (SWE-agent ACI) / 77.5% (Codex CLI) / 82.5% (Aider) / 85.0% (vendor tool-format); n=100; smallest p vs bare: `vendor tool-format` (p=0.004, higher than bare); tokens -16% to +68%	the accompanying release
`bfcl_parallel_multiple` (worst scaffold on this category)	89.5% (bare) / 92.5% (SWE-agent ACI) / 86.6% (Codex CLI) / 46.3% (Aider) / 88.1% (vendor tool-format); n=100; smallest p vs bare: `Aider` (p<0.001, lower than bare); tokens +8% to +62%	the accompanying release

C.5 Dose response of consensus selection

Consensus selection is a deterministic function of (corpus subset, threshold), so how much corpus it needs can be enumerated rather than sampled. We enumerate every subset of the corpus - all $C(7, m)$ for m=1..7, no cherry-picking - at six thresholds, and record whether the selected mechanism set equals the full-corpus set.

Scope note (deliberate). This enumeration runs over all 7 chains that were grown on this category, including the three that the exclusion policy of Appendix A.2 marks L0-hard-excluded. That is consistent rather than contradictory: the enumeration is a statement about the selection rule (how the mechanism set responds to corpus size and threshold), not a measurement of any chain's quality, and the published artifact was itself built from this same corpus. The previous appendix shows that rebuilding it on the policy-clean subset yields a behaviourally identical specification, so no number in this section depends on the choice.

m (chains)	subsets	mean components	saturation vs full corpus	distinct sets	modal share
1	7	4.25	0%	4	14%
2	21	5.50	0%	6	5%
3	35	3.37	49%	6	49%
4	35	3.88	63%	2	63%
5	21	3.71	71%	2	71%
6	7	4.00	100%	1	100%
7	1	4.00	100%	1	100%

Table: recurrence threshold 0.5 (the published artifact's value).

Threshold axis at full corpus. Components selected as the threshold loosens (1.0 / 0.857 / 0.714 / 0.6 / 0.5 / 0.34): 1.0 -> 1 / 0.857 -> 1 / 0.714 -> 3 / 0.6 -> 3 / 0.5 -> 4 / 0.34 -> 4. Tightening to unanimity collapses the artifact to the single mechanism that every chain proposed; loosening to 0.34 adds nothing, because the mechanism distribution is long-tailed: of 20 distinct proposed mechanisms, 12 appear in exactly one chain and 1 in all of them.

Reading. (i) The corpus saturates well below its full size: at m=4 a majority of subsets already select the full-corpus set, and from m=6 every subset does. (ii) The threshold is a relative regulariser: with $m \leq 2$ chains recurrence is trivially satisfied, so the same threshold selects more components than at m=7 - voting degenerates into unfiltered aggregation, which the main text measures as harmful. Consensus therefore needs enough chains for 'recurrence'

to mean something, but not many. (iii) With 12 singletons out of 20 mechanisms, consensus is not 'pick the common edit'; it is 'keep what a majority reached, with a floor'.

What this section does not measure. It is a statement about the selected mechanism set, not about accuracy: we did not evaluate each m variant on held-out tasks, because at the available clean-stratum sizes a single-variant comparison resolves only about ±3.7 pp (n=300, q≈0.05), well above the effect we are trying to track. The accuracy axis is therefore left open, deliberately.

C.5.1 The vote table, mechanism by mechanism

This table is the mechanism-level expansion of the selection rule of §3, computed over the same corpus the published artifact was built from (7 chains grown on `bfcl_live_multiple`, single-step). It backs the three properties the main text argues from: a long tail of singletons, a majority rule that is not sufficient, and cross-seed / cross-family recurrence for every survivor. One row per distinct coarse mechanism `op:kind@hook`. Each row reports how many of the 7 chains proposed it, and its recurrence. It also reports how many meta-model families and how many seed harnesses those chains span, whether that kind can ever execute in this environment, and whether the mechanism is in the shipped artifact. Rows are ordered by recurrence, and the four rows the rule selects are in bold.

mechanism	chains	recurrence	families	seeds	inert here	in artifact
`modify:system_prompt@task_start`	7/7	1.00	2	2	-	yes
`add:action_format@context_post`	5/7	0.71	2	2	-	yes
`add:decompose@task_start`	5/7	0.71	2	2	-	yes
`param:param@`	5/7	0.71	2	2	-	-
`add:tool_doc_augment@tool_pre`	4/7	0.57	2	2	-	yes
`modify:self_check@task_end`	4/7	0.57	2	2	yes	-
`add:answer_guard@task_end`	3/7	0.43	2	1	yes	-
`delete:decompose@`	2/7	0.29	1	1	-	-
`add:retry_policy@`	1/7	0.14	1	1	-	-
`add:retry_policy@control`	1/7	0.14	1	1	-	-
`add:self_check@task_end`	1/7	0.14	1	1	yes	-
`delete:error_reframe@`	1/7	0.14	1	1	-	-
`delete:retry_policy@`	1/7	0.14	1	1	-	-
`delete:self_check@`	1/7	0.14	1	1	yes	-
`modify:answer_guard@`	1/7	0.14	1	1	yes	-
`modify:answer_guard@task_end`	1/7	0.14	1	1	yes	-
`modify:history_limit@context_pre`	1/7	0.14	1	1	-	-
`modify:obs_format@context_pre`	1/7	0.14	1	1	-	-
`modify:system_prompt@`	1/7	0.14	1	1	-	-
`modify:tool_doc_augment@tool_pre`	1/7	0.14	1	1	-	-

Counting conventions on the accepted side (disclosed; not used in the main text). The artifact is a function of proposals only, so nothing above depends on how acceptance is read; we record both readings because in this corpus they differ, and a reader comparing our tables with the released logs should know which is which. Reading A takes the edit list archived with each acceptance decision, which is the list before the type contract filtered it: 15 distinct mechanisms, 28 chain-mechanism instances, including `modify:action_format@` that the vote table does not contain at all. Reading B is uniform over the corpus: the round-level final verdict (the three places the logs record it - rounds, gate decisions and the archived summary - agree round by round, and the check that produced this table aborts if they ever do not) crossed with the contract-filtered edit list: 13 distinct mechanisms, 27 instances. The entire difference is one round of one chain, in which two parameter-only edits were rejected by the type contract yet remain visible in that round's archived edit list. No claim in this paper is computed from either reading.

Self-check. The analysis that produced this table re-derives the recurrence histogram of the previous section and the composition of the published artifact by an independent counting path, verifies that the three places the logs record an acceptance verdict agree round by round, and aborts on any disagreement. Status in the release: `PASS`.

C.6 Stateful multi-turn integration and its power ceiling

This section exists because the strongest objection to a null result measured only on single-turn categories is that the components which could matter there (`retry_policy`, `history_limit`, `error_reframe`, `obs_format`) are inert by construction in a single-step environment. We therefore integrated the public stateful multi-turn category and ran the same three-point audit we require of every benchmark integration before testing any hypothesis.

Data and backends, verbatim from upstream. `multi_turn_base` (200 tasks, 1,142 ground-truth calls) plus the eight official backend classes and the official `multi_turn_checker` (state + response) are taken unchanged from the Gorilla repository; scoring is done only by the official checker, never by an equivalence rule of our own. Loading uses the official `_load_scenario(deepcopy(config))` path, which is the only route that initialises backend state (e.g. `cwd`) before scoring; instantiating a backend as `Class(**config)` leaves that state unset and silently zeroes every score, so the loader is called exactly as upstream calls it.

(1) Wiring self-check. Replaying the official ground truth through our own execution path yields 200/200 valid under the official checker. This is the check that separates 'the task is hard' from 'the integration is wrong'.

(2) Answer-channel audit. 19 official execution responses inspected: ['payload string'] --- no empty-string or 'false success' channel, i.e. the checker reads every channel through which an answer can arrive.

(3) Pool saturation / floor. At n=24, k=2 with the same frozen task model that scores 74-92% on single-turn BFCL categories:

arm	task success	tok/task	mean steps	client-error
`no_instructions`	4.17%	40,478	8.85	0%
`swe_agent_aci`	0.00%	61,410	12.42	0%

The failure mode is the opposite of saturation: a floor. Diagnosis per task shows every turn is reached (mean turns 3.67), the backends report zero execution errors, and the official checker fails with `['multi_turn:instance_state_mismatch']` - the agent acts, but not on the state the task requires. Mean per-turn diagnostic accuracy was 1.0 only because our own turn-level diagnostic is lenient; the official all-turn verdict is what counts.

Consequence for power (declared before running the arm comparison). At a floor near 4.2% success the paired discordance is necessarily tiny, so the achievable MDE is larger than on the single-turn categories, not smaller: taking $q \approx 0.10$, the MDE is about 18.1 pp at n=24, 7.7 pp at n=133 and 6.3 pp at n=200 (the entire pool). Resolving a 5 pp difference would need about 315 paired tasks - more than the pool contains. Per-task cost is also far higher: 40,478 tokens/task for the bare arm vs 1,431 on single-turn live_multiple (stratum A, the released logs).

What we therefore report. The stateful integration is verified (official checker, 200/200 GT replay) but undecidable at the power this pool and budget allow: no arm comparison on multi_turn_base can distinguish effects smaller than ~6 pp, and the frozen task model sits at a floor where the consensus artifact's ~1.7 pp effect is far below it. This is stated in the main text as a limitation (L1) rather than as evidence either way, and the arm comparison is listed as ongoing. Recording an uninformative experiment with its power arithmetic is, in our view, more useful than omitting it.

C.6.1 The four-arm comparison, actually run

Arms fixed in advance (`bare` = `no_instructions`, the transcribed SWE-agent ACI, the vendor tool-call template, and the frozen consensus artifact), n=45 held-out tasks, k=2, same frozen task model, official checker:

arm	task success (official checker)	95% CI	tok/task	mean steps	active	client-error
bare(no_instructions)	2.22%	[0.0, 6.7]	39,522	8.64	1	0%
swe_agent_aci	0.00%	[0.0, 0.0]	61,042	12.44	6	0%
vendor_toolfmt	0.00%	[0.0, 0.0]	41,991	8.23	3	0%
pop:consensus	0.00%	[0.0, 0.0]	42,304	8.06	4	0%

Paired deltas against the bare scaffold are all ≤ 0 with at most one discordant task per arm, i.e. nothing is distinguishable: this is a floor, not a ranking. Cost per task is 39,522 tokens for the bare arm (28x the single-turn price), so the arm comparison cost 16,637,308 tokens in total.

Decision, pre-declared and honoured. Since the pool cannot resolve an effect of our size (MDE ~7.7 pp at n=133, ~6.3 pp at n=200) and the frozen task model is at a floor, we report the stateful integration as verified but underpowered, list multi-turn as ongoing, and draw no directional conclusion from it. We did not keep increasing n until something became significant.

C.7 Which chain is behind each evolved arm

The arm names in the decisive 8-arm experiment are built from the last 14 characters of the chain run-id, which is not unique across chains. The mapping below is derived from the accompanying release at build time, and the build fails if the mean dev gain of the chains behind these arms stops equalling the `devGains` macro quoted in the main text.

Disclosure. `evolved:-no_instr-8873`: the last-14-character suffix is shared by `bflm-kimi-r0-no_instr-8873`, `bflm-kimi-r1-no_instr-8873`; under the loader order the later one wins. our analysis code now fails on such a collision instead of overwriting silently.

Mean dev gain over the corpus-A chains behind these arms = +6.67 pp, i.e. exactly the value quoted in the main text; the excluded chain `bflm-kimi-r0-no_instr-8873` (+0.83 pp dev) is not part of it and is not one of the two arms measured.

arm in the 8-arm experiment	chain directory	dev baseline	dev final	dev delta	corpus-A	accepted edits
evolved:-no_instr-8873	bflm-kimi-r1-no_instr-8873	71.67%	77.50%	+5.83 pp	yes	2
evolved:-swe_agen-8873	bflm-kimi-r0-swe_agen-8873	73.33%	80.83%	+7.50 pp	yes	2

C.8 Typed-surface expressiveness: the complete rejection audit

Across 24 logged chains the type contract rejected 16 of 165 proposed edits. Classification:

class	count	meaning
shape:params-not-nested	12	shape:params-not-nested
shape:missing-target	2	shape:missing-target
other	1	other
missing-kind	1	[zh 1ch sha d08d064a] kind [zh 2ch sha 49ecd0e3]

Key number: 15 of 16 (94%) are output-format errors that a single input normalizer would recover, and 0 rejections were for a mechanism outside the 13 declared kinds. In other words, the typed surface cost us essentially no expressiveness in practice; the binding constraint was the meta-model's patch format. Of the 129 kept edits, 33 (25.6%) changed free-form prose only (the paper sourcest, error_reframe.params.rules, tool_doc_augment` text), which is the channel through which a free-form surface would have worked anyway.

Every rejection, with the edit it proposed; Chinese rationales are masked to a length+sha pointer (byte-exact text in the released EvoLog):

chain	round	class	rejected edit	stated rationale (verbatim bytes in EvoLog)
lm-kimi-r1-no_instr-8873	4	shape:params-not-nested	modify:system_prompt@None	[zh 124ch sha 3b8ec200]
lm-kimi-r1-no_instr-8873	4	shape:params-not-nested	modify:action_format@None	[zh 91ch sha 6b65c5e2]
lm-kimi-r1-swe_agen-8873	3	shape:params-not-nested	modify:answer_guard@None	[zh 136ch sha f08e350e]
lm-kimi-r1-swe_agen-8873	3	shape:params-not-nested	modify:system_prompt@None	[zh 115ch sha 65328e56]
lm-kimi-r1-swe_agen-8873	3	shape:params-not-nested	add:retry_policy@None	[zh 94ch sha a4f0a588]
lm-qwen-r0-no_instr-8873	1	other	modify:system_prompt@None params=['style', 'text']	[zh 36ch sha a0cb3a70]
lm-qwen-r0-swe_agen-8873	2	shape:params-not-nested	modify:system_prompt@None	[zh 80ch sha 5b1ad42f]
lm-qwen-r0-swe_agen-8873	3	shape:params-not-nested	modify:system_prompt@None	[zh 71ch sha f2aea438]
lm-qwen-r0-swe_agen-8873	3	shape:params-not-nested	modify:tool_doc_augment@None	[zh 52ch sha fb9a275e]
g4demo	2	shape:params-not-nested	modify:action_format@None	[zh 62ch sha 049aeb9a]
g4demo	2	shape:params-not-nested	modify:tool_doc_augment@None	[zh 43ch sha feee2624]
g4demo	3	shape:missing-target	modify:answer_guard@None	[zh 72ch sha 206a66a2]
g4demo	3	missing-kind	modify:None@None	[zh 89ch sha ecd4b5c6]
p1c-kimi-r0-5365	1	shape:missing-target	modify:retry_policy@control params=['max_retries', 'on']	[zh 82ch sha 2c2146e8]
p1c-kimi-r1-5365	2	shape:params-not-nested	add:decompose@None	[zh 45ch sha 610f6b15]
p1c-kimi-r1-5365	2	shape:params-not-nested	add:self_check@None	[zh 35ch sha b690cb71]

D Robustness Checks and Sensitivity Analyses

D.1 Robustness checks, exclusions and measurement-validity disclosures

Exclusion policy. Chains are graded mechanically from EvoLog fields --- zero baseline tokens, no completed evolution round, a held-out batch failing the client-error guard, or a measured k=1 flip rate above 0.10 --- with the criteria declared before outcomes and the per-chain grades, the thresholds and the selected chain sets tabulated in Appendix A.2. On the public benchmark, 3 chains are hard-excluded from result-bearing analyses and 2 more are retained under disclosure for fingerprint and decision statistics only, never for a dev/held-out delta. Nothing is dropped silently: the excluded chains are named in that table together with the field that excluded them, and every corpus-level statistic in this paper is additionally reported under corpus-B, a second policy that applies no noise-based exclusion at all and changes none of its conclusions.

Determinism. Environments are a deterministic function of `(split, seed)`, and friction noise a deterministic function of `crc32(task_id) * 10 + sample_index`, so any environment in this paper can be regenerated exactly and diffed against the released one. Nothing depends on a process-dependent hash, on wall-clock order, or on the evaluation worker count: two regenerations of the same `(split, seed)` are byte-identical, and this is asserted by the released environment code rather than assumed.

Four checks every measurement in this paper passes. Each is cheap, and each is a sensitivity check a loop can run on logs it already keeps.

1. Evaluation-batch validity. Every evaluation batch carries a client-error rate. Above the declared guard the batch is retried and the evaluation then raises instead of returning a score, so a mass API failure cannot be recorded as a harness failure, enter the corpus, or decide a gate. One held-out batch in this corpus fails that rule and is excluded above, and the same guard is what excludes the two arms reported in Appendix D.2.
2. Static coverage of the edit surface. Every processor kind in the type system is dispatched by the frozen interpreter, and a static coverage analysis reports, per chain, the fraction of type-checked edits that target a component the environment never executes (12.6% of kept edits corpus-wide). This is the only instrument in the rig that can see such an edit: a gate reading the artifact or the score cannot.
3. Answer channel and saturation of every benchmark integration. A verifier must read every channel through which the policy can answer, and every benchmark is saturation-probed over its entire pool --- bare scaffold and best transcribed scaffold --- before any result is computed on it. MGSM scores 97.2% for the bare scaffold over all 250 items, so it has no headroom and serves as a no-regression probe only. A number computed on an integration that fails either check is not reported.
4. Intent claims as typed fields, not free text. What an edit claims to fix is declared as a typed field and matched against the environment's friction inventory: the gate acts on 1 of 147 edits with 0 false rejections, and in the end-to-end validation 9/9 edits carried a declaration, with none undeclared. This is exact because the claim is a field; a keyword classifier over an agent's prose rationale instead inherits an unmeasured error rate, and on this corpus that error rate is large enough to fabricate a finding.

Two measurement-validity disclosures. First, the flip-rate criterion is applied with a disclosure rather than uniformly: 2 retained chains measured k=1 flip rates (0.150/0.133) above the 0.10 threshold, and they are kept only because a within-seed statistic needs two chains per seed and this corpus has two seeds. Every quantity that involves them is also reported under corpus-B in Appendix A.2. Second, reported sample sizes are the realized ones. Four of the five public categories had development pools materially smaller than the 100 tasks requested, and the smallest realized pool is 67 tasks. That is why every n in this paper is read back from the log of the evaluation that produced it, not from the request.

Engineering hazards that silently corrupt results if unhandled. Each of the following is a property of the served models or the transport, not of our code. Each is documented in the supplement, and a self-evolution result that does not control for them is not a result. Reasoning models emit thinking outside `content`, and they return empty strings when `max_tokens` is small. Some model families do not support `response_format=json_object`. JSON truncation is not detectable by an emptiness check. Network errors must be retried, while 4xx must not. An evolver will invent placeholder names outside the template contract. Identical specifications do not give identical scores at temperature > 0. Reported sample sizes must be the realized ones.

D.2 Difficulty gradient and the two evaluation windows

The public benchmark's held-out task order is a difficulty gradient, so absolute scores at different n are not comparable and every row in this paper is paired inside one window. The two windows used here, and the arithmetic that relates them, are given below so that any window-level number can be checked against the pooled one.

quantity	value	source
bare accuracy, tasks 1-100 (window 1)	71.00%	the released logs per-task
bare accuracy, tasks 101-300 (window 2)	84.83%	same file
bare accuracy, all 300 (measured)	80.33%	the released logs
reconstructed from the two windows	80.22%	window-weighted mean
window-1 campaign (n=100), bare	68.33%	the accompanying release (client-error 7% / 18% on two arms)

The reconstruction differs from the measured 300-task score by 0.11 pp: a window-level number and a pooled number over the same list differ by exactly the prefix they cover, plus sampling noise.

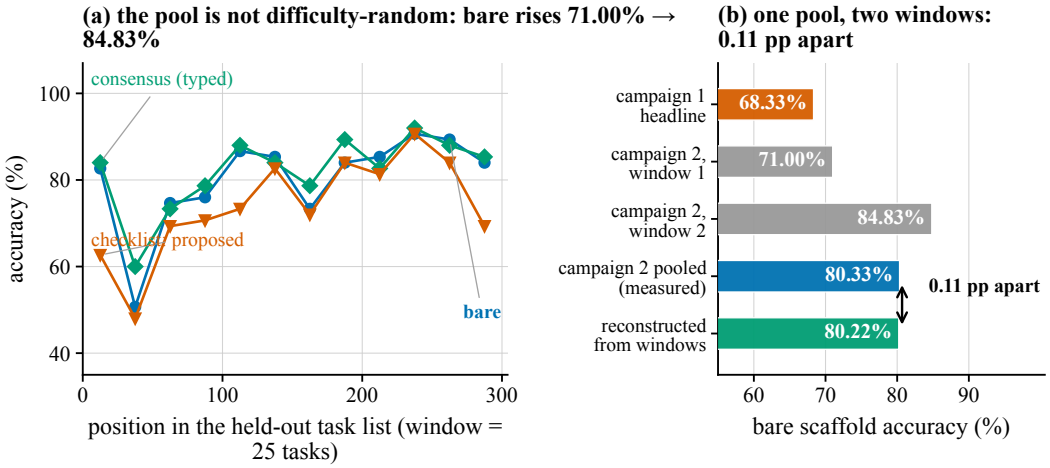


Figure 5: One pool, two windows. Window 1 = tasks 1--100, window 2 = the remainder of the same 300-task held-out list; the earlier window-1 campaign hit a 7% / 18% client-error rate on two arms, above the batch-validity guard, so those two arms are not used. Rolling accuracy over the held-out task order shows the benchmark is not difficulty-random, and the bar chart shows why a window-level number and the pooled number over the same list differ: window 1 is the harder prefix. Reconstruction arithmetic in the table above.

E Reproducibility Details

E.1 What the supplement contains, and how to re-derive this paper

The supplement is organized so that a reader can re-derive any number without reading our code: the released logs are the source of truth, and the analysis layer is a deterministic function of them.

- Fingerprints and corpus statistics. Every edit is reduced to a coarse fingerprint (`operator:kind@hook`) and a fine one (fingerprint plus parameters); cross-chain agreement is Jaccard over those sets with empty pairs excluded, and recurrence, component survival and checklist distillation are computed from the same fingerprints. These are the quantities behind §6.2.
- Statistics. Stratified bootstrap confidence intervals, exact McNemar tests, Wilcoxon signed-rank tests and AUROCs are implemented directly (no SciPy dependency) and unit-checked against hand-computed cases; every p-value in the paper is the exact test, not a normal approximation.

- Reconstruction. The released reconstruction check replays each chain's logged edit sequence from its seed specification and asserts equality with every archived intermediate harness version --- the guarantee that a chain's final arm really is the harness its log says it is.
- Noise instruments. The noise floor and the component ablation of §6 and §5.4 are produced by two small, deterministic re-scoring procedures over already-collected episodes, so neither costs new rollouts to audit.
- Exclusions. Inclusion grades (hard-excluded / disclosed-retained / delta-clean / sensitivity corpus) are derived from log fields by a single machine-graded policy, so no chain is ever named or excluded by hand, and the same policy is re-run inside the claim audit.
- One command. A single pipeline in the supplement recomputes all of the above from the released logs and rebuilds every table, figure and number in this paper; if recomputation changes a measurement, the paper changes with it.

E.2 Public benchmarks, transcribed harnesses and verifiers

ToolOrbit is self-built, so it cannot carry the paper alone. We therefore run the same loop on public benchmarks with their official verifiers, and seed the loop from published harnesses.

Benchmarks. BFCL v4 [Gorilla/BFCL, `bfcl_eval/data`] provides the scored categories we use: `live_multiple` (1 053 real user-contributed functions), `parallel_multiple` (200, requires several calls with several candidate functions), `parallel` (200), `multiple` (200), and `irrelevance` (240, where the correct behaviour is to call nothing --- a built-in negative control against call-spamming). Verification is BFCL's AST subset matching, reimplemented with recursive support for nested dict/list parameters and JSON-string coercion (unit-tested); dev/held-out splits are stride-interleaved with zero overlap, verified at load time. MGSM-en (250 problems, 83/167 split) serves as a cross-domain transfer probe: no business tools, one `submit_answer`, numeric exact match.

Transcribed harnesses. Each published harness is mapped onto our processor kinds and recorded with its source path and commit SHA (the provenance table below). We state plainly that this is a lossy transcription of design decisions, not the original code.

Table 0 --- Published harnesses on a public benchmark (task model Qwen3.8-27B, frozen; BFCL v4 `live_multiple`, n=100 per cell, single sample; tokens per task in parentheses; CIs are bootstrap 95%; McNemar is paired against the bare scaffold).

harness (source)	accuracy	95% CI	tok/task	McNemar p
`no_instructions` (SWE-agent's own shipped ablation `config/demo/the released configuration)	74.0%	[65.0, 82.0]	1 397	---
`swe_agent_aci` (SWE-agent `config/the released configuration` + tools/windowed + windowed_edit_linting + review_on_submit_m`)	79.0%	[71.0, 87.0]	1 642 (+18%)	0.359
`codex` (Codex CLI the accompanying notes + `prompts/templates/compact`)	80.0%	[72.0, 88.0]	1 483 (+6%)	0.180
`aider` (Aider our analysis code, architect/editor + lint-fix)	75.0%	[66.0, 83.0]	1 871 (+34%)	1.000
`vendor_toolfmt` (vendor function-calling template)	75.0%	[66.0, 83.0]	1 553 (+11%)	1.000

Table 0b --- The same five harnesses across four public BFCL v4 categories and MGSM-en (task model Qwen3.8-27B frozen; single sample per task; per-cell n is capped by each category's dev pool --- `live_multiple` n=100, `irrelevance` n=80, `multiple` n=67, `parallel_multiple` n=67, MGSM n=84; bootstrap 95% CI; tokens/task in parentheses; paired McNemar against the bare scaffold).

harness	BFCL irrelevance (n=80)	BFCL live_multiple (n=100)	BFCL multiple (n=67)	BFCL parallel_multiple (n=67)	MGSM-en (n=84)
no_instructions (bare)	73.8% [64,82] (1077)	74.0% [65,82] (1397)	94.0% [88,99] (819)	89.5% [82,96] (1020)	96.4% [91.7,100] (683)
swe_agent_aci	75.0% [65,84] (1348)	79.0% [71,87] (1642)	94.0% [88,99] (1022)	92.5% [85,99] (1301)	96.4% [91.7,100] (890)
codex	77.5% [68,86] (1014)	80.0% [72,88] (1483)	94.0% [88,99] (877)	86.6% [78,94] (1108)	97.6% [94.0,100] (689)
aider	82.5% [74,90] (1814)	75.0% [66,83] (1871)	95.5% [90,100] (1287)	46.3% [34,58] (1658)	97.6% [94.0,100] (1082)
vendor_toolfmt	85.0% [76,92] (900)	75.0% [66,83] (1553)	97.0% [93,100] (923)	88.1% [81,96] (1147)	98.8% [96.4,100] (672)

Power caveat, stated where it belongs: at n=67 the MDE is ±8.9 pp and at n=84 it is ±8.0 pp (§6). The `multiple` / `parallel_multiple` / MGSM cells can therefore only support claims about effects larger than that. That is exactly why we report the −43 pp Aider gap on `parallel_multiple` (far above the floor) and treat the sub-5 pp differences as noise rather than a ranking. The Aider gap on `parallel_multiple` is the one effect in this table that sits far above that floor. The `live_multiple` arm (n=100) is re-measured at k=3 in Table 6.

Significance (paired McNemar vs bare scaffold, n=100).

benchmark	harnesses significantly better	significantly worse	verdict
BFCL live_multiple	none ($p \geq 0.18$)	none	scaffolds indistinguishable
BFCL multiple	none	none	near ceiling (94--97%)
BFCL irrelevance	vendor_toolfmt p=0.004, aider p=0.039	none	format discipline helps refraining from spurious calls
BFCL parallel_multiple	none	aider p<0.001 (46.3% vs 89.5%)	a published scaffold is significantly harmful here
MGSM-en	none (all $p \geq 0.5$; at most 2 discordant tasks)	none	saturated (96.4--98.8%)

Four readings, all of which are used later in the paper:

1. A published scaffold can be significantly harmful. `aider` (architect/editor split + 2-round self-check + memory cheatsheet) collapses to 46.3% on `parallel_multiple` where the bare scaffold gets 89.5% (p<0.001). Multi-step deliberation machinery applied to a single-turn parallel-call decision actively destroys performance --- and at +63% tokens. This is the strongest available argument for per-domain harness adaptation, and it is why we evolve from two very different seeds (Q1').
2. Where scaffolds differ significantly, it is on behavioural discipline, not on reasoning. The only BFCL improvement that survives testing is on `irrelevance` (+11.2 pp, p=0.004) for the harness whose distinguishing feature is strict tool-call formatting.
3. The bare scaffold is the cheapest arm on all four BFCL categories (819--1 397 tok/task vs up to 1 871 for `aider`) and is never significantly worse on any of the five benchmarks. Cost-accuracy Pareto domination by the empty scaffold holds on 4 of 5 benchmarks; on MGSM the cheapest arm is `vendor_toolfmt` (672 vs bare's 683 tok/task) at 98.8% --- a 2.4 pp, non-significant difference for 1.6% fewer tokens. This is the empirical seed of §6.3.
4. Headroom is benchmark-specific: 3 pp on `multiple`, 7.5 pp on `parallel_multiple`, 15 pp on `irrelevance`, 20 pp on `live_multiple`, 2.8 pp on MGSM (97.2% over all

250 items --- saturated). We therefore run evolution on `live_multiple` (largest public headroom with a non-degenerate baseline) and use MGSM strictly as a no-regression probe (Appendix A.5): it cannot show gains, only harm.

The `irrelevance` category is also our public-benchmark instrumentation for reward-hacking propensity: 15--26% of failures there are spurious calls --- an agent acting when acting is wrong. Any evolved harness that lowers that number has learned restraint; any harness that raises it has learned to look busy.

Difficulty is a design requirement, not an accident. Calibration is part of the environment's specification: without the friction layer the bare seed harness solves the environment outright, zero headroom leaves evolution nothing to do, and every reported gain would be noise. The friction layer therefore exists to leave measurable headroom while keeping the environment oracle-solvable, and the unfrictioned and frictioned baselines are both reported as part of that specification.

E.2.1 Transcription and data provenance

artifact	source	revision
`sweagent_default`	https://github.com/SWE-agent/SWE-agent/blob/main/config/the released configuration	
`sweagent_no_instructions`	https://github.com/SWE-agent/SWE-agent/blob/main/config/demo/the released configuration	
`sweagent_windowed`	https://github.com/SWE-agent/SWE-agent/blob/main/tools/windowed/the released configuration	
`sweagent_edit_linting`	https://github.com/SWE-agent/SWE-agent/blob/main/tools/windowed_edit_linting/the released configuration	
`sweagent_review_on_submit`	https://github.com/SWE-agent/SWE-agent/blob/main/tools/review_on_submit_m/the released configuration	
`codex_gpt5_codex_prompt`	https://github.com/openai/codex/blob/main/codex-rs/core/the accompanying notes	
`codex_base_instructions`	https://github.com/openai/codex/blob/main/codex-rs/protocol/the accompanying release	
`codex_compact`	https://github.com/openai/codex/blob/main/codex-rs/prompts/templates/compact/the accompanying notes	
`aider_base_prompts`	https://github.com/Aider-AI/aider/blob/main/aider/coders/our analysis code	

E.2.2 Published scaffolds on public benchmarks, cell by cell

scaffold	Live multiple (n=100)	Multiple (n=67)	Parallel multiple (n=67)	Irrelevance (n=80)	MGSM-en (n=84)
bare (reference)	74.0% (1397 tok)	94.0% (819 tok)	89.5% (1020 tok)	73.8% (1077 tok)	96.4% (683 tok)
SWE-agent ACI	+5.0 (1.18×)	+0.0 (1.25×)	+3.0 (1.28×)	+1.2 (1.25×)	+0.0 (1.30×)
Codex CLI	+6.0 (1.06×)	+0.0 (1.07×)	-3.0 (1.09×)	+3.7 (0.94×)	+1.2 (1.01×)
Aider	+1.0 (1.34×)	+1.5 (1.57×)	-43.3* (1.63×)	+8.7* (1.68×)	+1.2 (1.58×)
vendor tool-fmt	+1.0 (1.11×)	+3.0 (1.13×)	-1.5 (1.12×)	+11.2* (0.84×)	+2.4 (0.98×)

Δ accuracy in pp vs `bare`, token multiple in parentheses; * = exact McNemar $p < 0.05$ against `bare` on that pool, at the n printed in the column head. No transcribed scaffold separates from `bare` on MGSM-en (saturated) or on three of four BFCL categories; where output formats are brittle, scaffold choice dominates by tens of points. The table is computed from the probe results in the released logs; the pools are the ones named in the column heads, each evaluated at its own realized n.

E.3 Seed harnesses, verbatim prompts and the consensus artifact

The five seed harnesses, including the four transcribed from published systems. Prompts are reproduced verbatim, exactly as the frozen runtime renders them.

Bilingual note. Our meta-models were reachable in a bilingual configuration, so some intermediate evolved components are in Chinese; those are reproduced verbatim because they are the experimental artifact, and the

released EvoLog contains them unchanged. Every artifact used for a headline number in this paper --- the consensus specification and both distilled checklists --- is in English, and the four transcribed published harnesses are reproduced in their original English.

E.3.1 Seed: No Instructions

Exact identifier in the released corpus:

no_instructions

Active components: 1. Attribution:

SWE-agent no_instructions.yaml ablation (transcribed)

```
system_prompt@task_start: {
 "text": "",
 "style": "concise"
}
```

E.3.2 Seed: Swe Agent Aci

Exact identifier in the released corpus:

swe_agent_aci

Active components: 6. Attribution:

SWE-agent ACI (transcribed)

```
system_prompt@task_start: {
 "text": "You are an autonomous agent that solves software/business tasks
by inspecting state before acting. Think step by step and explain your
reasoning before each action. Do not guess identifiers or values:
open/query the relevant record first. Keep changes minimal and verify the
effect of every mutation by re-reading the state. Before submitting, review
each requirement in the task and confirm it is satisfied.",
 "style": "verbose"
}

obs_format@context_pre: {
 "render": "json",
 "truncate_chars": 3000
}

history_limit@context_pre: {
 "keep_last": 10,
 "summarize": false,
 "keep_first_n": 2
}

error_reframe@step_post: {
 "template": "[zh 5ch sha aad5587c]:{err}\n[zh 4ch sha dca7a118]:{hint}",
 "include_hint": true,
 "max_len": 300
}

self_check@task_end: {
 "rounds": 1,
 "checklist_driven": true
}

answer_guard@task_end: {
 "require_all_subgoals": true
}
```

E.3.3 Seed: Codex

Exact identifier in the released corpus:

codex

Active components: 5. Attribution:

Codex CLI harness (transcribed)

system_prompt@task_start: {
 "text": "You are a software engineering agent running in a sandbox with tools. Be precise and concise; avoid unnecessary actions. Prefer the smallest change that satisfies the request. When context grows long, rely on the summarized history rather than raw transcripts. If a tool call fails, inspect the error and adjust; do not repeat an identical failing call.",
 "style": "concise"
}

history_limit@context_pre: {
 "keep_last": 8,
 "summarize": true,
 "keep_first_n": 2
}

action_format@context_post: {
 "schema_hint": true,
 "strict_json": true,
 "repair": true
}

error_reframe@step_post: {
 "template": "[zh 4ch sha f190fa44]:{err}\n[zh 10ch sha 9a96fcb1],[zh 7ch sha 4f8d8a32]",
 "include_hint": true,
 "max_len": 250
}

self_check@task_end: {
 "rounds": 1,
 "checklist_driven": false
}

E.3.4 Seed: Aider

Exact identifier in the released corpus:

aider

Active components: 5. Attribution:

Aider architect/editor + lint-fix (transcribed)

system_prompt@task_start: {
 "text": "First act as an architect: decompose the task into a short plan and identify what must change. Then act as an editor: apply one change at a time. After each change, run the lint/test step and fix any reported problem before continuing.",
 "style": "checklist"
}

decompose@task_start: {
 "n_subgoals": 3,
 "explicit_checklist": true
}

error_reframe@step_post: {
 "template": "[zh 5ch sha b8af885a]:{err}\n[zh 9ch sha d907c194]:{hint}",
 "include_hint": true,
 "max_len": 300
}

self_check@task_end: {
 "rounds": 2,
 "checklist_driven": true
}

memory_cheatsheet@context_pre: {
 "capacity": 5,
 "write_on": [
  "success",
  "failure"
 ]
}

E.3.5 Seed: Vendor Toolfmt

Exact identifier in the released corpus:

vendor_toolfmt

Active components: 3. Attribution:

vendor tool-calling template (transcribed)

```
system_prompt@task_start: {
 "text": "[zh 8ch sha 8af1ccc0] agent[zh 12ch sha c779a70f];[zh 9ch sha
02d5d8b5],[zh 5ch sha 58ac826a]",
 "style": "concise"
}

action_format@context_post: {
 "schema_hint": true,
 "strict_json": true,
 "repair": true
}

tool_doc_augment@tool_pre: {
 "add_examples": false,
 "n_examples": 1,
 "warn_constraints": true
}
```

E.3.6 The consensus artifact (frozen; evaluated on every clean stratum)

```
{
 "version": "consensus",
 "parent": null,
 "max_steps": 2,
 "model_params": {
  "temperature": 0.3,
  "max_tokens": 2048
 },
 "note": "consensus>=0.50, 4 [zh 2ch sha d0f3e329]",
 "processors": [
  {
   "id": "system_prompt-a91f7e",
   "kind": "system_prompt",
   "hook": "task_start",
   "params": {
    "text": "You are a function-calling agent. Match the number of calls to
the number of distinct actions the instruction actually requests: if it
asks for several things, issue ALL corresponding calls before finishing (in
parallel within one step when possible); if it asks for one thing, make
exactly one call. When every required parameter/ID is explicitly given in
the instruction, call the target function DIRECTLY with exactly those
values; never add optional parameters the user did not request. Only when a
required identifier is missing may you issue ONE minimal lookup to obtain
it, then immediately proceed to the target call(s). Before finishing,
verify each requested action has been performed — no more, no less.",
    "style": "concise"
   },
   "enabled": true,
   "origin": "consensus"
  },
  {
   "id": "action_format-4d7d40",
   "kind": "action_format",
   "hook": "context_post",
   "params": {
    "schema_hint": true,
    "strict_json": false,
    "repair": false
   },
   "enabled": true,
   "origin": "consensus"
  },
  {
   "id": "decompose-9adf3d",
   "kind": "decompose",
   "hook": "task_start",
   "params": {
    "n_subgoals": 0,
    "explicit_checklist": true
```

```
    },
    "enabled": true,
    "origin": "consensus"
   },
   {
    "id": "tool_doc_augment-f54b08",
    "kind": "tool_doc_augment",
    "hook": "tool_pre",
    "params": {
     "add_examples": false,
     "n_examples": 0,
     "warn_constraints": true
    },
    "enabled": true,
    "origin": "consensus"
   }
  ]
}
```


Vote table (mechanism → number of chains that proposed it):


```
{
 "n_runs": 7,
 "min_recurrence": 0.5,
 "votes": {
  "modify:system_prompt@task_start": 7,
  "delete:decompose@": 2,
  "param:param@": 5,
  "delete:error_reframe@": 1,
  "delete:retry_policy@": 1,
  "add:tool_doc_augment@tool_pre": 4,
  "add:decompose@task_start": 5,
  "add:action_format@context_post": 5,
  "modify:tool_doc_augment@tool_pre": 1,
  "modify:obs_format@context_pre": 1,
  "modify:system_prompt@": 1,
  "add:retry_policy@": 1,
  "modify:history_limit@context_pre": 1,
  "add:retry_policy@control": 1
 },
 "kinds": [
  "system_prompt",
  "action_format",
  "decompose",
  "tool_doc_augment"
 ]
}
```