LATENT HARNESS: COMPILING AGENT BEHAVIORS INTO POST-BLOCK ACTIVATION SHIFTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Harness engineering and self-evolving agents improve language-model agents quickly, yet their gains live outside the model: the harness is re-sent and re-paid on every call, and the standard route to internalization is a weight update that is costly to iterate and fixed once merged. We propose compiling the harness instead. Latent Harness distills an external condition, either an explicit process-level behavior instruction or the behavioral contract of an agentic harness, into a rank-8 activation shift applied at the output of every decoder block of a frozen backbone (0.03–0.05% of parameters). At inference the compiled bundle carries no harness text, adds zero tokens, and exposes three runtime controls: injection strength, injection phase, and shift-space mixing. Compiled behaviors recover the source-harness gain in 8 of 10 preregistered cells, and where the explicit source pays the heaviest per-call token tax the compiled form surpasses it outright; they transfer zero-shot across tasks (5 of 7 pairs positive), compose in shift space while stacked text instructions interfere destructively, and lift budget-matched harness search by 1.45×. On three agentic benchmarks the same operator aligns policy to the harness contract: under matched data, capacity, and effective strength it outperforms same-capacity LoRA, which needs 11.1× the parameters to reach the same band, and it regresses nothing where the base policy is already strong. Across 1B-to-8B models the method skeleton is invariant while gain locations move with the model-task pair, which delimits where compiled harnesses should be deployed. Our code is available at `https://anonymous.4open.science/r/latentharness-C574`.

1 INTRODUCTION

The performance of a language-model agent is increasingly set by its harness: the instructions, tool policies, memory, and verification scaffolds wrapped around model calls. Recent work searches harnesses over program spaces (Hu et al., 2025; Zhang et al., 2025b), modular design spaces (Shang et al., 2025; Zhang et al., 2025a), and safety-constrained evolutionary spaces (Zheng et al., 2025); another line makes harnesses improve themselves, synthesizing code harnesses from failures (Lou et al., 2026), editing executable runtime harnesses (Shao et al., 2026), or optimizing harness and model jointly (Lee et al., 2026). Surveys document how broad this engineering surface has become (Meng et al., 2026). The gains are real. Where they end up, however, is almost always one of two places: in text that surrounds the model, or in the model's weights.

Both carry structural costs. A textual condition is a recurring expense: even a 49-token process instruction is re-sent on every call, the behavior exists only while the text is present, and serving several harnesses multiplies prompt length. Weight updates internalize at a different price. Training-style policy alignment such as SFT, RLOO, and GRPO (Shao et al., 2024), co-evolution frameworks that update policy weights alongside the harness (Chen et al., 2026c;b; Hebbar et al., 2026), and self-evolving agents that write acquired skills into fine-tuned or RL-trained weights (Huang et al., 2026; Xia et al., 2026; Zhang et al., 2026b) all modify millions of parameters per behavior, bind one behavior to one set of edited weights, and fix the intervention strength at merge time.

We propose a third destination: compile the harness. At compile time the frozen model produces its own supervision under the external condition, as teacher traces generated with an explicit behavior instruction prepended, or as decision pairs harvested from trajectories that succeeded under an agen-

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.

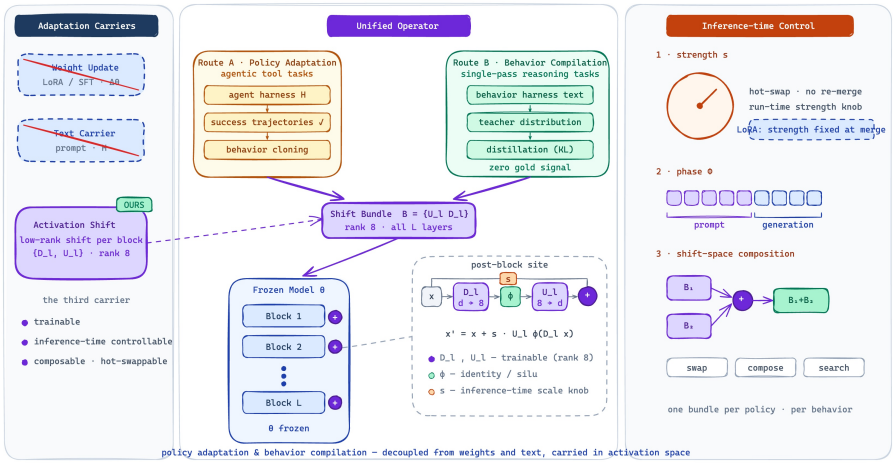


Figure 1: Latent Harness compiles external conditions into post-block activation shifts. Teacher traces or strictly filtered success trajectories train a rank-8 shift at every block output of a frozen backbone; the bundle injects at run time with strength, phase, and mixing controls.

tic harness. A rank-8 low-rank shift is then trained at the output of every decoder block and exported as a per-layer bundle. At run time the bundle is injected into the residual stream after each block: the model behaves as if the condition were present, without the condition text, at zero additional tokens, with every backbone parameter bit-identical. We call the compiled artifact a *latent harness*. Figure 1 summarizes the pipeline.

Compilation preserves behavior to a degree we did not anticipate. Across preregistered cells, the compiled shift recovers most of the explicit-harness gain, and in the cells where the text harness pays the heaviest per-call token tax the compiled form surpasses it, because the internalized behavior pays that tax once, at compile time. The compiled object is the behavior rather than generic task skill: preregistered behavioral markers move with the scores, accuracy-only adapters dissociate from behavior, shuffled pairings fail to reproduce it, and a harmful harness compiles into a harmful latent with the same fidelity. Compiled bundles are reusable objects: they transfer zero-shot across tasks, compose in shift space while stacked text instructions interfere destructively, and lift budget-matched harness search by 1.45×. The same operator aligns policy on three agentic benchmarks: under matched data, capacity, and effective strength it outperforms same-capacity LoRA, and on a benchmark where the base policy is already strong it regresses nothing. Our contributions are:

- **A compilation abstraction for harness gains.** One operator, a post-block rank-8 shift on a frozen backbone (0.03–0.05% of parameters), compiles two kinds of external conditions into an artifact that needs no condition text and no tokens at inference, with runtime strength, phase, and mixing controls.
- **Evidence that compilation preserves the behavior.** 8 of 10 preregistered cells recover at least half of the source gain, attributed by a three-chain protocol over scores, preregistered markers, and controls, including symmetric replication of harmful signatures; where the text source pays the heaviest token tax, the compiled form surpasses it outright.
- **Evidence that compiled bundles are reusable.** Zero-shot cross-task transfer (5 of 7 pairs positive, 59–74% out-of-distribution recovery), exact composition algebra in shift space where text stacking is destructive, and 1.45× search gain at a matched evaluation budget.
- **Policy compilation on agent benchmarks.** At 1.012× parameters and matched data and strength, the shift beats same-capacity LoRA on ToolHop (60.00% vs. 54.74%, paired net +5); full-capacity LoRA needs 11.1× (4B) and 3.09× (8B) parameters to reach the same band; API-Bank shows zero regression; the skeleton is invariant from 1B to 8B while gain move with the model-task pair.

2 Compiling Behaviors into Post-Block Shifts

2.1 The Post-Block Injection Site

Let the frozen policy be an autoregressive transformer π_θ with L decoder blocks. In residual-stream notation, block l computes $x^{l+1} = x^l + f_l(x^l)$. We inject at the *post-block* site, the point where

everything block l computed meets everything blocks $l{+}1, \ldots, L$ will read. For every layer we train a down projection $D_l \in \mathbb{R}^{r\times d}$ and an up projection $U_l \in \mathbb{R}^{d\times r}$ with rank $r{=}8$, and replace the block output by

$$\tilde{x}_i^{l+1} \;=\; x_i^{l+1} \;+\; s\,\alpha\, U_l\big(\varphi(D_l\, x_i^{l+1})\big)\,\mathbf{1}[i \in \Phi], \tag{1}$$

where s is a global runtime strength, α is a fixed training scale, φ is a bottleneck nonlinearity, and the phase set Φ selects which token positions receive the shift. Trainable parameters total $2rdL$, between 0.029% and 0.042% of the backbone across the five models we study (Table 3 in the appendix).

Two properties make this site the right compilation target. First, a perturbation of the residual trunk is read by every subsequent layer, so one parameter influences far more computation than a perturbation inside a single linear map, which propagates only within its layer. Second, the site is the unique junction at which any first-order correction to a block's behavior can be expressed, which is why two very different compile sources succeed under the same operator. Adila et al. (2026) show that weight updates can be viewed as activation shifts; we compile conditions that never required a weight update into the same substrate. The operator is first-order kin to LoRA (Hu et al., 2021): both inject rank-at-most-r perturbations, which is what makes capacity-aligned comparison possible (Section 3.2). The bottleneck nonlinearity is $\varphi = \mathrm{SiLU}$ with zero-initialized U_l for instruction compilation and the identity for policy compilation; both choices work, indicating low sensitivity to this detail.

2.2 Compile source A: explicit behavior instructions

The first source is a registry of seven process-level behavior instructions, each a short (at most 60 tokens), single-pass, tool-free prompt prefix that targets a diagnosed error mode: independent verification, numbered decomposition, write-the-expression-first evaluation, concise answering, option elimination, evidence quoting, and test-interface consultation. Each entry is registered with a frozen hash, carries a predicted behavioral marker (a deterministic parse or regex count on the raw output, such as decomposition-step count or interface-alignment rate), and enters the study only if screening shows a significant explicit gain (Appendix B).

The teacher is the frozen model with the instruction prepended; it produces greedy traces that are cached in full and filtered by correctness. The student is the same frozen model plus the shift of Eq. 1, trained on prompts that are *byte-identical* to the unconditioned base prompts, so the instruction text never appears in any student input (asserted per batch). The distillation loss is token-level cross-entropy on filtered teacher traces for generation tasks and an option-level KL on cached teacher answer distributions for multiple-choice tasks. No gold labels enter this arm. Two capacity-matched controls share the budget: a gold-label task adapter and a joint objective (Appendix A).

2.3 Compile source B: harness-conditioned strategies

The second source is the behavioral contract of an agentic harness. The frozen base policy rolls out on the training pool under the official harness pairing; each trajectory decomposes into per-step decision pairs whose prompt is the full interaction history rendered so far and whose completion is the action taken. The *strict tier* keeps pairs only from trajectories whose final answer is correct. The motivation is the choice of target distribution for behavioral cloning (Torabi et al., 2018): failed trajectories are draws from the base policy itself, and training on them pulls the target back toward the base policy. We observed exactly this: a balanced tier costs 3.2 accuracy points and inflates action counts by 51%, while the strict tier gains 3.2 points with 3% inflation. Training applies prefix-masked cross-entropy to action tokens with hyperparameters identical to the LoRA comparison arm (Appendix A).

2.4 Runtime interface

Compilation exports a bundle $\{U_l D_l\}_{l=1}^{L}$ in fp32 per layer; load-back is bit-exact and the engine's weights are untouched. Three controls operate at inference without retraining: the global strength s in Eq. 1, hot-swappable within a serving session; the phase $\Phi \in \{\text{prompt}, \text{generation}, \text{both}\}$; and shift-space mixing, where two compiled bundles combine as $\alpha Z_a + \beta Z_b$ realized exactly by a rank-16 merged bundle (tensor-level error at most 10^{-5}). Mixing must be defined on induced shifts rather than factor sums, because the bottleneck nonlinearity makes factor-space addition the wrong

Table 1: Main results on the HarnessForge benchmark suite, following its baseline suite. Search-style rows search over agent harnesses; training-style rows update policy weights; HarnessForge is the full co-evolution framework. **Ours** evaluated under our controlled single-round alignment protocol (official round-3 harness pairing, held-out pools; protocol-distinct from the co-evolution rows above). 4B ToolHop: two independent Ours runs reproduce 55/95 exactly.

		ToolHop		SearchQA			API-Bank	
Model	Method	Ans.	Path	Hotpot	2Wiki	Overall	Succ.	API.
Qwen3-4B	ADAS	40.00	44.99	28.67	28.00	28.33	57.31	60.99
	AgentSquare	29.23	39.46	29.33	30.00	29.67	47.82	46.10
	AFlow	31.28	43.81	30.67	29.33	30.00	45.72	41.13
	MaAS	42.05	51.80	30.67	30.00	30.33	62.74	63.83
	MermaidFlow	44.62	58.74	31.33	28.00	29.67	64.33	60.28
	SFT	45.13	63.90	36.67	38.67	37.67	75.10	73.76
	RLOO	46.15	64.87	35.33	40.00	37.67	78.82	76.60
	GRPO	49.74	66.41	36.00	**43.33**	39.67	76.46	75.18
	HarnessForge	52.82	68.10	**42.00**	42.00	42.00	80.07	82.27
	Ours	**57.89**	**71.30**	40.00	42.00	41.00	76.47	81.82
Qwen3-8B	ADAS	42.05	61.32	31.33	32.00	31.67	60.14	59.57
	AgentSquare	30.77	41.22	29.33	30.00	29.50	54.37	53.90
	AFlow	32.82	54.12	31.33	32.67	32.00	49.74	47.52
	MaAS	44.62	51.80	32.00	32.00	32.00	66.72	60.28
	MermaidFlow	47.39	59.31	32.67	30.67	31.67	67.46	62.41
	SFT	48.72	72.20	40.67	41.33	41.00	73.80	73.05
	RLOO	50.77	72.75	40.00	39.33	39.67	73.76	71.63
	GRPO	51.28	75.62	41.33	42.00	41.67	74.71	73.76
	HarnessForge	54.87	74.05	41.33	44.33	42.83	78.22	78.01
	Ours	**62.46**	**76.55**	**42.67**	**50.67**	**46.70**	67.65	72.73

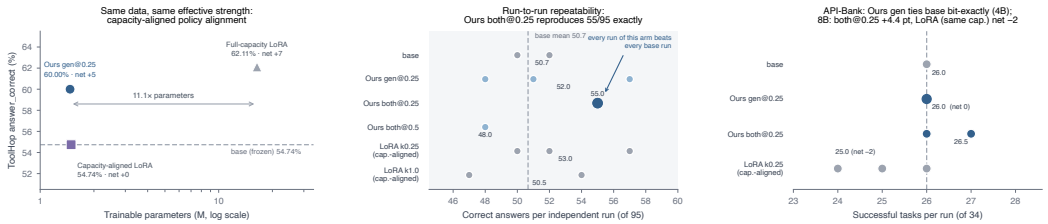
algebra (Section 3.5 demonstrates the consequences). Two guards keep measurements honest: if metrics are bit-identical across two strengths the injection is silently off, which we check per arm; and a bounded, arm-independent parse protocol repairs or fail-closes malformed tool-call JSON identically for every arm.

3 EXPERIMENTS

We organize the evaluation around four questions. Does policy compilation work on agentic benchmarks, and at what parameter cost (Section 3.2)? Does instruction compilation preserve the explicit harness's gain (Section 3.3)? What exactly is compiled (Sections 3.4 and 3.5)? And where does the intervention act, and when does it fail (Section 3.6)?

3.1 SETUP

Instruction compilation uses Llama-3.2-1B-Instruct (M1), Qwen3-4B (Yang et al., 2025) (M2), Llama-3.1-8B-Instruct (M3), and an independent replication track on Llama-3.2-1B (base), evaluated on BoolQ (Clark et al., 2019), ARC-Challenge (Clark et al., 2018), GSM8K (Cobbe et al., 2021), and MBPP (sandboxed pass@1), with SVAMP and ARC-Easy as out-of-distribution targets. Policy compilation uses Qwen3-4B and Qwen3-8B on ToolHop (Ye et al., 2025) (n=95), SearchQA over HotpotQA (Yang et al., 2018) and 2WikiMultihopQA (Ho et al., 2020) (n=50), and API-Bank (Li et al., 2023b) (n=34), all under official round-3 harness pairings with held-out task pools disjoint from training. Statistics follow one discipline: preregistered criteria, selection on validation only, a single test-set touch, paired bootstrap 95% CIs (B=10,000) over 3 training seeds for instruction cells, and at least 2 independent full evaluations per adopted policy arm with per-item paired net differences. Appendix A gives the full protocol.



(a) Capacity-aligned comparison **(b)** Run-level repeatability **(c)** API-Bank zero regression

Figure 2: Policy compilation: efficiency, stability, safety. (a) Capacity alignment on ToolHop. (b) Run-level repeatability. (c) Zero regression on API-Bank at 4B and 8B.

3.2 Policy compilation on agentic benchmarks

Table 1 reports policy compilation on the HarnessForge benchmark suite. Under our single-round protocol (official harness pairing, held-out pools, unaligned base policy, at least two repetitions per arm), the compiled shift alone reaches 57.89% answer accuracy on ToolHop at 4B and 62.46% at 8B, above the published full co-evolution numbers on the same backbones (52.82 and 54.87), and lands within a few points of them on SearchQA and API-Bank. The dagger rows differ in protocol from the co-evolution rows above them, so we read them as context rather than a head-to-head comparison.

Capacity-aligned comparison. We align three variables at once: the LoRA control acts on q/k/v with rank 3 and 1.49M parameters (1.012× ours), trains on the same 829 ToolHop decision pairs, and applies the same effective intervention multiplier (0.5). Figure 2a shows the outcome: the shift reaches 60.00% (paired net $+5$ over base) while the capacity-aligned LoRA nets zero, and full-capacity LoRA (rank 8, all linear projections, 16.4M parameters, 11.1× ours) reaches 62.11%. At 8B, LoRA r11 (3.09× our parameters) matches the best shift arm (62.63% vs. 62.46%). The efficiency gap follows from the site: a post-block parameter is read by every downstream layer, an in-layer parameter is not. The shift also stays scorable at full strength (s=1.0, net $+1$), whereas merged LoRA deltas at enlarged coefficients produced no scorable outputs under the earlier protocol; the runtime strength knob doubles as a contract-preserving safety margin.

Stability. Agentic evaluation under greedy decoding still forks run to run, so single runs mislead. The `both@0.25` configuration reproduced 55/95 correct in two independent runs, an exact aggregate match despite 14 per-item flips, and it is the only arm whose every run exceeds every base run (Figure 2b). Its path score (0.713) is the highest of all arms, and it triggered zero fail-closed events: prompt-phase injection does not strain the tool-call contract. Gains also come without action inflation (14.41 steps vs. base 11.80 and full-capacity LoRA 14.61).

Zero regression where base is strong. On API-Bank, a short-chain, format-heavy benchmark where the base policy is already strong, the generation-phase shift ties base bit-for-bit across three base runs (0.7647) while the capacity-aligned LoRA nets -2 with all three runs below base (Figure 2c). At 8B every shift arm is at or above base and `both@0.25` adds $+4.4$ points. Preserving an already-good policy is a property weight updates did not provide at matched capacity.

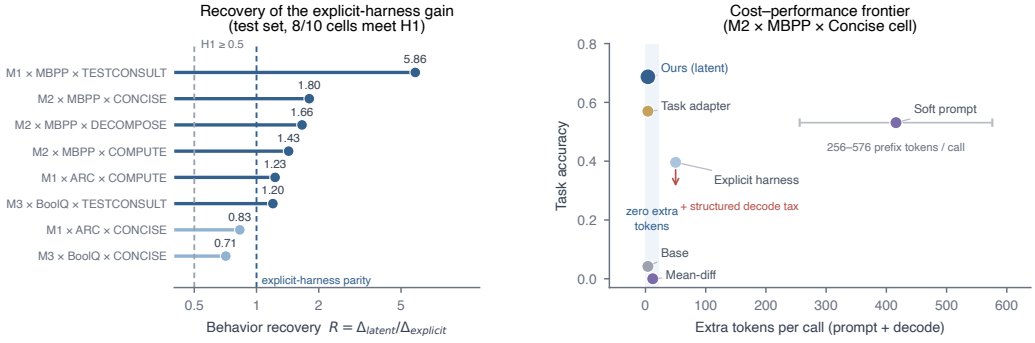
3.3 Instruction compilation preserves the source gain

Screening first establishes honest denominators: of 49 (harness × model × dataset) cells, 10 show a significant explicit gain and enter the study (Appendix B). Table 2 reports the preregistered controlled test-set comparison, and Figure 3a plots the recovery rate $R = \Delta_{\text{latent}}/\Delta_{\text{explicit}}$: eight of the ten cells meet the preregistered criterion ($R \geq 0.5$ with the 95% CI excluding zero).

Three results stand out. First, the flagship cell distills M1's test-consult behavior on MBPP from only 20 correct teacher traces (teacher pass@1 is 0.073 at the format floor) into $+31.7$ test points. Recovery ratios divide by the explicit gain, so cells with a small explicit source produce large ratios (R=5.86 here, against an explicit gain of $+5.4$ points); we therefore read this cell by its absolute delta, and the observation we defend is that the information needed to install a behavior is far smaller than the information needed to solve the task. Second, on M2×MBPP the compiled form lands $+19$

Table 2: Controlled test-set comparison (one touch, preregistered). SOFT-PROMPT is exactly parameter-matched and trained on the same distillation signal; TASK-ADAPTER is trained on gold labels at matched capacity. Δ_L: ours over base; $R = \Delta_L/\Delta_E$, with explicit gains in Fig. 7. All Δ_L CIs exclude zero; 8 of 10 enrolled cells meet the criterion $R \geq 0.5$.

Cell	EXPL.	SOFT-PROMPT	TASK-ADAPTER	**OURS**	Δ_L (R)
M1 · ARC × COMPUTE	0.300	0.347	0.533	**0.322**	+0.082 (1.23)
M1 · ARC × CONCISE	0.380	0.376	0.533	**0.362**	+0.122 (0.83)
M1 · MBPP × TESTCONSULT	0.066	0.079	0.308	**0.323**	+0.317 (5.86)
M2 · MBPP × DECOMPOSE	0.410	0.508	0.562	**0.661**	+0.619 (1.66)
M2 · MBPP × COMPUTE	0.470	0.521	0.554	**0.660**	+0.618 (1.43)
M2 · MBPP × CONCISE	0.396	0.531	0.570	**0.687**	+0.645 (1.80)
M3 · BoolQ × CONCISE	0.809	0.833	0.891	**0.774**	+0.077 (0.71)
M3 · BoolQ × TESTCONSULT	0.807	0.822	0.893	**0.826**	+0.128 (1.20)



(a) Recovery across cells **(b)** Token cost vs. performance

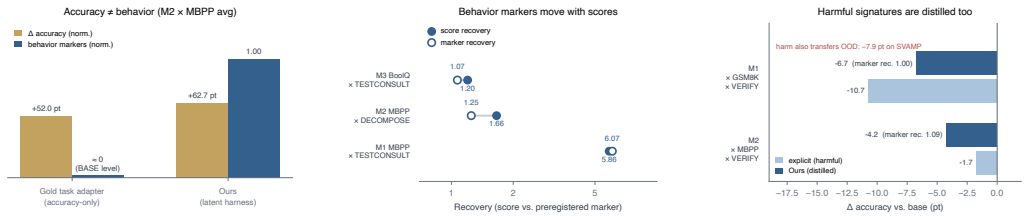
Figure 3: Instruction compilation. (a) Behavior recovery on the test set. (b) Token cost against accuracy.

to +29 points above its explicit source (0.660–0.687 vs. 0.396–0.470), and there the denominator is not small: the explicit harness itself gains +0.36 to +0.43 over a near-zero base. The mechanism is the token tax: the explicit harness spends decode budget at every call, while the compiled behavior spends it once at compile time and nothing per call, which also places Latent Harness alone at the zero-token point of the cost-performance frontier, far from parameter-matched soft prompts that pay 256–576 tokens on every call (Figure 3b). Third, the pattern replicates on an independent track with a base (non-instruct) 1B model, recovering the source gain on both cells (Appendix D).

3.4 What is compiled: the behavior, not task skill

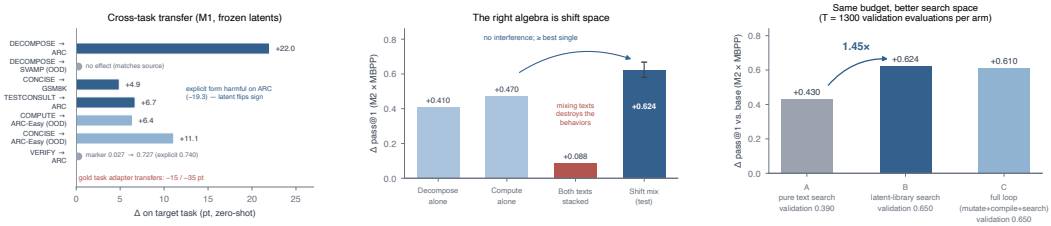
Accuracy alone cannot attribute a compiled artifact, because a gold-label adapter trained at matched capacity is the *strongest* accuracy arm (up to +0.65) while leaving every behavioral marker at base level (Figure 4a). We therefore preregistered three evidence chains. The score chain is Table 2. The marker chain shows behavior moving with scores: decomposition steps rise from 0 to 4.6 (explicit source 4.5), interface alignment from 0.039 to 0.97, answer-line rate from 0.525 to 1.00, each matching score recovery in magnitude (Figure 4b). The control chain closes the alternatives: shuffling teacher traces across harnesses flattens every cell and every marker, and exactly parameter-matched soft prompts trained on the same distillation signal are weaker everywhere they can be compared while paying hundreds of tokens per call.

The sharpest test of specificity is symmetric: if the bundle encodes harness semantics, harmful harnesses should compile as faithfully as helpful ones. They do. A verification harness that hurts M1 on GSM8K (−10.7 points explicit) distills to −6.7 points with verify-action marker recovery 1.00, and a harmful M2 pairing reproduces −4.2 with marker recovery 1.09; the negative signature even transfers out of distribution to SVAMP (−7.9 points, Figure 4c). Whatever the bundle carries, it carries the harness's behavior, not a generic improvement.



(a) Accuracy vs. behavior **(b)** Marker recovery **(c)** Harmful signatures

Figure 4: Three-chain attribution. (a) Accuracy against behavior markers (M2×MBPP average). (b) Score recovery against marker recovery. (c) Harmful signatures compile symmetrically and transfer out of distribution.



(a) Zero-shot transfer **(b)** Composition algebra **(c)** Budget-matched search

Figure 5: Reuse of compiled behaviors. (a) Cross-task transfer of frozen bundles. (b) Text stacking against shift-space composition. (c) Search gain at matched evaluation budgets.

3.5 Compiled bundles are reusable objects

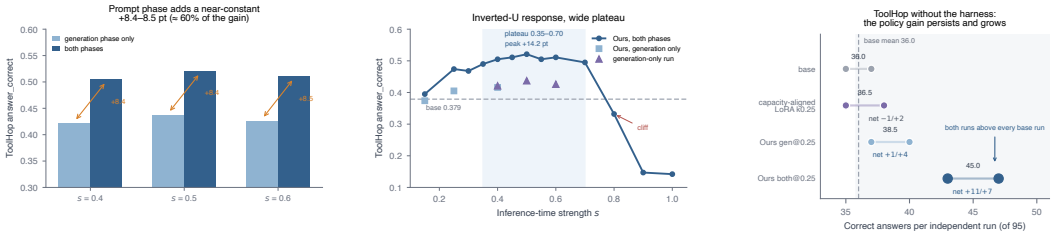
Transfer. Freezing five compiled bundles and evaluating seven preregistered (source → target) pairs with no retraining and no target-side tuning, five pairs transfer positively with CIs excluding zero (Figure 5a). Decomposition transfers to ARC with $+22.0$ points and the marker follows (steps $1.10 \rightarrow 5.20$). Two out-of-distribution rows into ARC-Easy recover 59% and 74% of their source gains. One transfer flips the sign usefully: the test-consult harness is harmful on ARC in text form (-19.3) yet its compiled form gains $+6.7$, because compilation distills the behavior as the teacher expresses it under that task rather than the instruction's literal text. The verification bundle transfers no score on ARC at all but moves its marker from 0.027 to 0.727 (explicit form: 0.740), pure behavior migration. Gold-label task adapters, by contrast, transfer at -0.15 and -0.35: task skills overfit, compiled behaviors do not.

Composition. Two compiled behaviors for M2×MBPP (decompose, compute) combine exactly in shift space through the rank-16 merge; every nonzero point of a 25-point (α, β) validation grid lands in 0.60–0.67 against a base of 0.02, and the frozen test combination reaches $+0.624$, no worse than the best single component with both marker signatures intact. Stacking the same two instruction texts in context is destructive: $+0.088$, a near-total collapse caused by mixed compliance with two mandated structures. The right algebra for behaviors is the shift space, not the text space.

Search. If bundles compose, a library of them turns harness search from discrete text rewriting into continuous coefficient selection. With evaluation budgets pinned at exactly $T{=}1300$ validation evaluations per arm on M2×MBPP, text-space search reaches $+0.430$ test gain, latent library search reaches $+0.624$, and the full loop (text mutation, compilation, coefficient search) reaches $+0.610$: the compiled library alone contributes the $1.45\times$ multiplier (Figure 5c).

3.6 Where the intervention acts, and when it fails

The compiled policy gain is not the harness's shadow. The policy gain does not require the harness to be present. With the harness removed and the protocol otherwise unchanged, `both@0.25` still delivers $+9.5$ points on ToolHop (paired net $+11/+7$; both runs above every base run), while the capacity-aligned LoRA stays at base level (Figure 6c). Removing the harness enlarges rather



(a) Phase decomposition **(b)** Strength response **(c)** Without the harness

Figure 6: Mechanism. (a) Phase decomposition of the gain. (b) Strength response with plateau and cliff. (c) ToolHop with the harness removed, by arm.

than shrinks the policy gain, indicating partial substitution: the shift performs structural corrections that the harness would otherwise impose, and it does so for behaviors the harness text does not cover.

The action site is the representation layer. Prompt-phase injection changes every downstream computation through attention and the KV cache, while generation-phase injection changes token-level dynamics only. Decomposing the two (Figure 6a): prompt positions contribute $+8.4$ to $+8.5$ points, near-constant across strengths and about 60% of the total. Instruction compilation is the limiting case of the same fact: the student trains on byte-identical prompts and still acquires the behavior entirely through representation-level correction.

Strength has a wide safe plateau and a sharp cliff. Sweeping 13 strength values with two repetitions each (Figure 6b) gives an inverted U with a 0.35–0.70 plateau in which all seven points fall within 2.6 points of each other, peaking at $+14.2$ over base. Beyond the plateau the failure is specific and diagnosable: trajectories complete normally, but `has_valid_answer` collapses monotonically ($1.000 \rightarrow 0.284$), a termination failure driven by prompt-side over-injection; generation-only injection never triggers it at any tested strength. The optimal strength is condition-dependent (0.25 with the harness, 0.35–0.70 without), which is exactly why it is exposed as a runtime knob rather than baked into weights.

What survives a scale change. Re-running the identical pipeline on Qwen3-8B preserves the skeleton: zero-weight adaptation works, the parameter-efficiency ordering holds (LoRA needs $3.09\times$ parameters to match), the failure family is the same with an earlier onset, and API-Bank remains regression-free. What moves is the gain location: ToolHop gains spread across several strengths instead of one, and SearchQA flips from no gain at 4B to $+7.5$–9.2 points at 8B (Appendix I). Applicability is therefore established per model-task pair, and the bundle's runtime controls are what make per-pair deployment practical.

4 Related Work

Harness engineering and self-evolution. Automated harness design searches programs (Hu et al., 2025; Zhang et al., 2025b), modular pipelines (Shang et al., 2025; Zhang et al., 2025a), and safety-constrained evolutionary spaces (Zheng et al., 2025); self-improving systems synthesize or edit harnesses from execution feedback (Lou et al., 2026; Shao et al., 2026; Zhang et al., 2026a; Lin et al., 2026a; Karten et al., 2026), optimize harness and model end-to-end (Lee et al., 2026), or compile harness logic into code and skill programs (Ning et al., 2026; Liu et al., 2026; Chen et al., 2026a). Self-evolving agents typically internalize what they learn through weight updates: RL-trained reasoning from zero data (Huang et al., 2026), recursive skill-augmented RL (Xia et al., 2026), memory skills (Zhang et al., 2026b), and self-referential improvement loops (Yin et al., 2025); Lin et al. (2026b) disentangle harness updates from harness benefits. HarnessForge (Chen et al., 2026b) co-evolves harnesses and policies and provides the benchmarks, harness pairings, and training protocol we adopt; its training-style arms (SFT, RLOO, GRPO) and co-evolution objective all write gains into weights. Latent Harness is complementary: it moves a fixed harness's behavioral content into activations, keeps the backbone frozen, and leaves the harness-evolution loop free to iterate on text.

Parameter-efficient adaptation. LoRA (Hu et al., 2021), prefix-tuning (Li & Liang, 2021), and prompt tuning (Lester et al., 2021) attach low-rank or prompt-side parameters to a frozen backbone.

Our operator shares the low-rank parametrization but acts at the post-block junction of the residual stream, and our comparisons hold data, capacity, and effective strength fixed to isolate the site rather than the budget. Behavioral cloning (Torabi et al., 2018) and data-efficient reasoning distillation (Zhang et al., 2026c) are output-mimicking relatives of ours; the differences are strict correctness filtering of the source and a preregistered behavior-level attribution protocol on the target.

Activation steering and representation engineering. Adding direction vectors to activations (Turner et al., 2024; 2023), contrastive activation addition (Rimsky et al., 2024), representation engineering (Zou et al., 2023; Bartoszcze et al., 2025), inference-time intervention (Li et al., 2023a), function vectors (Todd et al., 2024), and learned representation interventions (Wu et al., 2024) study how latent-space edits change model behavior. Systematic evaluations find difference-vector methods unreliable and learned interventions stronger (Tan et al., 2024; Wu et al., 2025; Stoehr et al., 2024), with extensions to refusal steering (Sheng et al., 2026), defense (Zou et al., 2024), subspace models (Zhao et al., 2025), and hallucination detection (Park et al., 2025). Our work follows the learned-intervention regime. Adila et al. (2026) formalize weight updates as activation shifts; we use the post-block shift as a compilation target for external conditions and add what steering work has not addressed: behavior-level attribution, composition algebra, transfer, and agent-benchmark policy alignment under controlled capacity.

5 Limitations

Gain locations move with the model-task pair: ToolHop gains concentrate or spread across strengths, and SearchQA flips direction from 4B to 8B. We therefore claim per-pair applicability, not universal gains, and the runtime knobs exist to support per-pair tuning. Strength has a hard ceiling: beyond the plateau the policy stops producing valid answers, the failure is prompt-side specific, and the onset moves earlier with scale, so deployments must sweep s before fixing it. Our policy results are single-round alignment of a base policy to a fixed harness, not multi-round co-evolution, so Table 1 dagger rows are protocol-distinct from the published co-evolution numbers. Evaluation pools are small (95, 50, and 34 items); we mitigate with repeated full evaluations and per-item paired analysis, but absolute deltas of a few points should be read under that lens. Instruction compilation covers short process-level instructions; harnesses too complex to state as text are handled by the trajectory route.

6 Conclusion

We treated the harness as a compilation source rather than a prompt to repeat or a skill to fine-tune. A rank-8 shift at the post-block junction of a frozen backbone compiles both explicit behavior instructions and agentic-harness strategies into bundles that need no condition text, no tokens, and no weight change, while exposing strength, phase, and mixing at run time. Compiled behaviors recover and sometimes exceed their sources, transfer and compose in shift space, align policy at a fraction of LoRA's parameters under matched conditions, and fail in a diagnosable, bounded way. The compilation view turns harness engineering from a per-call expense into an asset that appreciates: every harness the community evolves becomes a candidate bundle in a reusable library.

Reproducibility statement

All numbers here come from preregistered protocols with fixed seeds, validation-only selection, single test touches, and append-only result ledgers; policy arms average at least two independent full evaluations. Appendix A details the harness registry, training configurations, arm definitions, parse guard, and statistical procedures; Appendix I documents cross-scale runs and protocol notes.

Ethics statement

Compilation is semantically faithful in both directions: harmful harnesses distill into harmful bundles as readily as helpful ones, which we report as an attribution finding rather than a capability. Deployment should registry-check compile sources, preregister evaluation criteria, and audit exported bundles; a practical advantage of the runtime form is that a bundle can be inspected, tuned down, or removed without touching model weights.

REFERENCES

Dyah Adila, John Cooper, Alexander Yun, Avi Trost, and Frederic Sala. Weight updates as activation shifts: A principled framework for steering, 2026. URL https://arxiv.org/abs/2603.00425.

Lukasz Bartoszcze, Sarthak Munshi, Bryan Sukidi, Jennifer Yen, Zejia Yang, David Williams-King, Linh Le, Kosi Asuzu, and Carsten Maple. Representation engineering for large-language models: Survey and research challenges. *arXiv preprint arXiv:2502.17601*, 2025.

Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. Skvm: Revisiting language vm for skills across heterogenous llms and harnesses. *arXiv preprint arXiv:2604.03088*, 2026a.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems, 2026b. URL https://arxiv.org/abs/2606.01779.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-harness: Co-evolving harnesses and model weights for llm agents. *arXiv preprint arXiv:2607.22688*, 2026c.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv. org/abs/2110.14168*, 9, 2021.

Prannay Hebbar, Yogendra Manawat, Samuel Verboomen, Alesia Ivanova, Selvam Palanimalai, Kunal Bhatia, and Vignesh Baskaran. Sia: Self improving ai with harness & weight updates. *arXiv preprint arXiv:2605.27276*, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in neural information processing systems*, 36:41451–41530, 2023a.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: Long papers)*, pp. 4582–4597, 2021.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026a.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, et al. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents. *arXiv preprint arXiv:2605.30621*, 2026b.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing llm agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P Murphy. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, et al. Agent harness for large language model agents: A survey. 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

Seongheon Park, Xuefeng Du, Min-Hsuan Yeh, Haobo Wang, and Yixuan Li. Steer llm latents for hallucination detection. *arXiv preprint arXiv:2503.01917*, 2025.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, 2024.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Shuai Shao, Kangning Zhang, Qingyao Li, Shijian Wang, Hao Wang, Wenxiang Jiao, Yuan Lu, Yi Guo, Weiwen Liu, and Weinan Zhang. Harness-r1: Learning to edit executable runtime harnesses from agent failure trajectories. *arXiv preprint arXiv:2608.02276*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Leheng Sheng, Changshuo Shen, Weixiang Zhao, Junfeng Fang, Xiaohao Liu, Zhenkai Liang, Xiang Wang, An Zhang, and Tat-Seng Chua. Alphasteer: Learning refusal steering with principled null-space constraint. In *International Conference on Learning Representations*, volume 2026, pp. 25861–25904, 2026.

Niklas Stoehr, Kevin Du, Vésteinn Snæbjarnarson, Robert West, Ryan Cotterell, and Aaron Schein. Activation scaling for steering and interpreting language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8189–8200, 2024.

Daniel Tan, David Chanin, Aengus Lynch, Brooks Paige, Dimitrios Kanoulas, Adrià Garriga-Alonso, and Robert Kirk. Analysing the generalisation and reliability of steering vectors. *Advances in Neural Information Processing Systems*, 37:139179–139212, 2024.

Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron Wallace, and David Bau. Function vectors in large language models. In *International conference on learning representations*, volume 2024, pp. 17282–17333, 2024.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. 2024.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Reft: Representation finetuning for language models. *Advances in Neural Information Processing Systems*, 37:63908–63962, 2024.

Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D Manning, and Christopher Potts. Axbench: Steering llms? even simple baselines outperform sparse autoencoders. *arXiv preprint arXiv:2501.17148*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, et al. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2995–3021, 2025.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Table 3: Models and coverage. Rank-8 post-block adapters, all layers, all tokens; SiLU bottleneck for instruction compilation, linear for policy compilation. Backbone frozen (per-parameter checksum verified).

Code	Model	Layers $\times$ d	Trainable	Share
M1	Llama-3.2-1B-Instruct	16×2048	524,288	0.0424%
M2	Qwen3-4B	36×2560	1,474,560	0.0367%
M3	Llama-3.1-8B-Instruct	32×4096	2,097,152	0.0312%
v2	Llama-3.2-1B (base)	16×2048	524,288	0.0424%
–	Qwen3-8B (policy)	36×4096	2,359,296	0.0288%
Instruction tasks: BoolQ, ARC-Challenge, GSM8K, MBPP (OOD: SVAMP, ARC-Easy)				
Policy benchmarks: ToolHop (n=95), SearchQA-hotpotqa (n=50), API-Bank (n=34)				

Lechen Zhang, Yunxiang Zhang, Wei Hu, and Lu Wang. Skill-aware data selection and fine-tuning for data-efficient reasoning distillation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 595–604, 2026c.

Haiyan Zhao, Heng Zhao, Bo Shen, Ali Payani, Fan Yang, and Mengnan Du. Beyond single concept vector: Modeling concept subspace in llms with gaussian distribution. In *International Conference on Learning Representations*, volume 2025, pp. 82998–83026, 2025.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*, 2025.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023.

Andy Zou, Long Phan, Justin Wang, Derek Duenas, Maxwell Lin, Maksym Andriushchenko, Rowan Wang, Zico Kolter, Matt Fredrikson, and Dan Hendrycks. Improving alignment and robustness with circuit breakers. *Advances in Neural Information Processing Systems*, 37:83345–83373, 2024.

A Experimental protocol details

Harness registry. Seven single-pass, tool-free, process-level instructions (VERIFY, DECOMPOSE, COMPUTE, CONCISE, ELIMINATE, QUOTE, TESTCONSULT), each at most 60 tokens, registered by sha256 hash with a named target error mode and preregistered markers computed by deterministic parsing of raw outputs (verify-action rate, decomposition-step count, arithmetic-expression rate, answer-line rate, option-coverage rate, evidence-quote rate, interface-alignment rate, clean-stop rate). Markers never fire on base-prompt echo by construction and are checked to that effect.

Instruction compilation. Teacher traces are greedy generations of the frozen model with the harness prepended; all outputs are cached, and correctness filtering selects the imitation target at training time. The student trains on $n{=}\min(600, |\text{pool}|)$ examples for 2 epochs at lr $9{\times}10^{-4}$ with 3 seeds $\{42, 43, 44\}$. Losses: prompt-masked token cross-entropy on filtered traces (generation) or option-level KL against cached teacher distributions (multiple choice), with option-string mirroring locked by unit test. Hard per-step assertions: backbone checksum equality, optimizer membership (adapter parameters only), and byte-level leakage checks (no harness substring in any student input; harness text present in every teacher prompt).

Policy compilation. Strict-tier decision pairs from harness-conditioned rollouts (ToolHop 1,042 pairs, 829 train / 213 val; SearchQA 670; API-Bank 327). Training: prefix-masked cross-entropy on completions, lr 10^{-4}, 3 epochs, batch size 1 with gradient accumulation 2, max length 4,096, bf16 with gradient checkpointing. The LoRA comparison arms use identical hyperparameters; the capacity-aligned arm is rank 3 on q/k/v (1.49M parameters, 1.012$\times$) with effective multiplier $\frac{\alpha_L}{r_L} k = 0.5$ matched to the shift's $s\,\alpha = 0.5$; the full-capacity arm is rank 8 on all linear projections (16.4M); at 8B the reference arm is rank 11 (7.30M, 3.09$\times$).

Evaluation. Greedy decoding throughout. Instruction cells: dataset-native extract-and-match or sandboxed pass@1 with preregistered extraction; paired bootstrap 95% CIs, B=10,000. Policy runs: HarnessForge evaluator fields used as produced, at least 2 independent full evaluations per adopted arm, per-item paired net differences, and a bounded deterministic parse guard (repair once, then fail-closed) that is identical across arms; fail-closed counts are reported as behavioral signals. Injection validity is verified per arm by the scale-signature check (bit-identical metrics across strengths indicate silent non-injection).

Arm definitions (instruction track). BASE (no intervention); EXPLICIT (harness text prepended, the recovery denominator); SOFT-PROMPT (exactly parameter-matched soft prefix trained on the same distillation signal); TASK-ADAPTER (gold-label cross-entropy, capacity matched); OURS (distillation without gold labels); plus SHUFFLED (teacher traces paired with the wrong harness) as the semantic-ablation control.

B EXPLICIT-HARNESS SCREENING AND THE MBPP FORMAT FLOOR

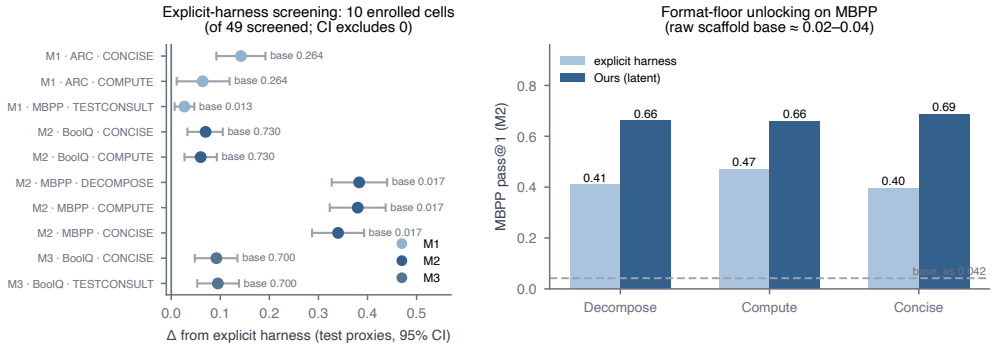


Figure 7: Screening forest (left) and the MBPP format-floor unlock (right).

Screening evaluates 49 (harness × model × dataset) cells with three validation draws each and admits the 10 cells whose explicit gain is positive with a 95% CI excluding zero (Figure 7, left, where each interval is one admitted cell). Cells whose explicit form does not help are recorded as not informative and never enter a recovery claim, by preregistration; the recovery rate is therefore never computed against a dead denominator. The admitted cells concentrate where the base model has a diagnosable failure mode rather than where the task is hardest. On MBPP all three models sit at a format floor (base pass@1 near 0.01–0.04: the 3-shot scaffold almost never elicits extractable code), and structural harnesses lift M2 to 0.34–0.42 while the corresponding markers move by design, decomposition steps $0 \rightarrow 4.2$ and interface alignment $0.057 \rightarrow 0.646$ (Figure 7, right). M3 passes a compliance pre-check with all three predicted markers moving in their predicted directions (evidence quotes $0.285 \rightarrow 0.755$, steps $0.09 \rightarrow 2.03$, verify actions $0.035 \rightarrow 0.980$), which is what makes the distillation targets of Section 3.3 unambiguous. Reading the forest by model also previews the recovery pattern: explicit gains are largest exactly where the base is weakest, so the flagship recoveries in Table 2 come from the hardest starting points rather than easy cells.

Figure 8 shows the two marker views that complement the score tables. The left panel tracks marker displacement under explicit harnesses in the MBPP family, confirming that the structural instructions move behavior first (steps, interface alignment, single-task blocks) and scores follow; the right panel shows the same quantities for the compiled form against base, with the explicit source as a reference point. The compiled markers reach or exceed the explicit references (steps 4.6 vs. 4.5, interface alignment 0.97, single-task blocks 1.00), which is the marker-level statement of the recovery result: the behavior is present, not merely a score advantage.

C ATTRIBUTION DETAILS AND PER-MODEL DISCUSSION

Table 2 compresses three model-specific stories. On M1, the two ARC cells recover the explicit gain (ratios 0.83 and 1.23 on moderate denominators), while the MBPP cell adds +31.7 points from only

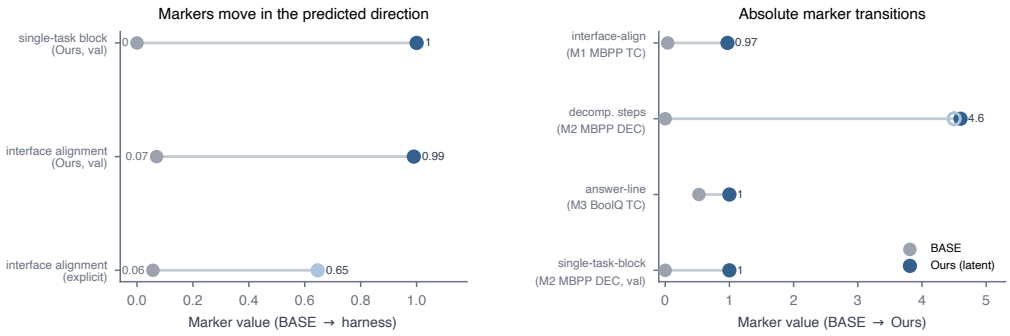


Figure 8: MBPP marker displacements under explicit harnesses (left) and absolute marker transitions from base to the compiled form (right).

20 correct teacher traces; its ratio of 5.86 reflects the $+5.4$-point explicit denominator rather than a fivefold effect, so we read the cell by its absolute delta. The exactly parameter-matched soft-prompt control trained on the same signal reaches 0.079 and pays 512 tokens per call, so the distillation signal is necessary and the activation site is what makes it sufficient. On M2, all three MBPP cells exceed their explicit sources (0.660–0.687 vs. 0.396–0.470): the explicit harness spends decode budget on mandated structure at every call within a 512-token limit, while the compiled form spends that budget on code. On M3, the two BoolQ cells bracket the criterion: concise answering recovers 0.71 of a large explicit gain (0.809 $\rightarrow$ 0.891 explicit, ours 0.774), while test-consult recovers 1.20 with 98% validation recovery and a seed range of 0.015, the most stable cell in the study. Across all cells the TASK-ADAPTER control is the strongest accuracy arm and the weakest behavioral arm, the dissociation that motivates the three-chain protocol in Section 3.4.

D RECOVERY SUMMARIES AND INDEPENDENT REPLICATION

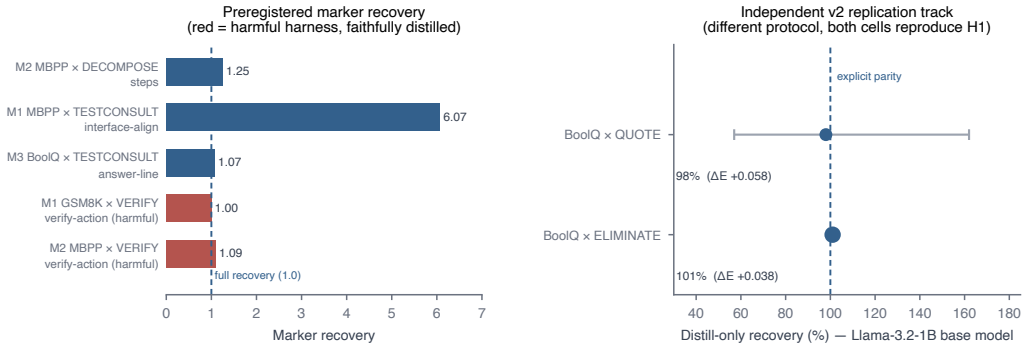


Figure 9: Preregistered marker recoveries across cells (left); independent replication on the base 1B model (right).

Figure 9 (left) collects marker recoveries across cells; the highlighted entries are the two harmful harnesses, whose verify-action behavior is reproduced at recovery 1.00 and 1.09 while their scores go negative. A generic task adaptation cannot produce a faithfully negative behavioral signature, which is why we read this panel as the single most discriminating attribution display in the study. The right panel reports the replication track: the full pipeline re-run on Llama-3.2-1B (base, non-instruct), an independent protocol with the complete BoolQ test set (n=654) and three training seeds. Both enrolled cells replicate, QUOTE at 98% recovery (CI $[+57, 162]\%$) and ELIMINATE at 101%, so the effect does not depend on instruct tuning or on one model family.

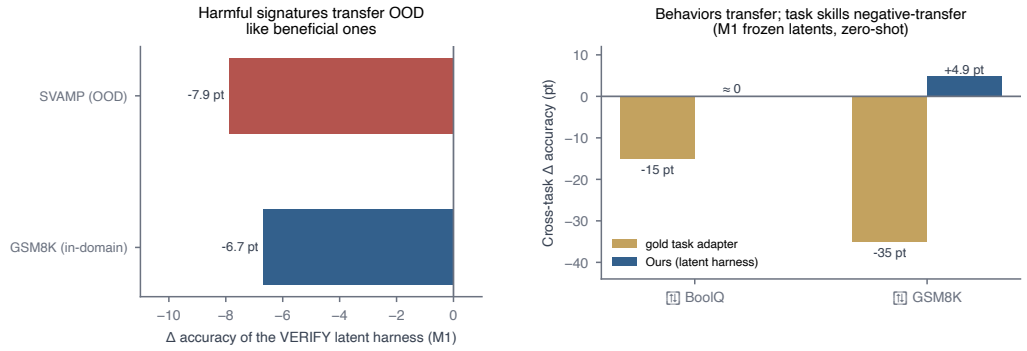


Figure 10: Harmful-signature transfer to SVAMP (left); transfer contrast between compiled behaviors and gold-label task adapters (right).

E HARMFUL-SIGNATURE TRANSFER AND THE TASK-SKILL CONTRAST

Two observations close the attribution argument. First, the harmful VERIFY signature is mobile: distilled from a GSM8K pairing at -6.7 points, it transfers out of distribution to SVAMP at -7.9, the same mobility positive signatures show. Harm and help moving through the same channel is the strongest available evidence that the bundle encodes harness semantics rather than task utility. Second, the transfer profiles differ in kind (Figure 10, right): task adapters trained on gold labels transfer negatively (-0.15 to BoolQ, -0.35 to GSM8K) while compiled behaviors transfer at zero or positive gain. Bundles behave like reusable behaviors; task adapters behave like overfit task skills.

F COMPOSITION AND SEARCH DETAILS

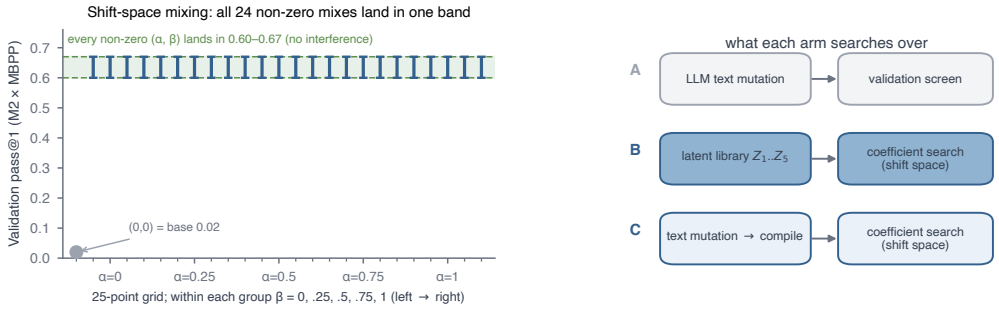


Figure 11: The 25-point shift-space composition grid (left); the three budget-matched search arms (right).

The composition grid (Figure 11, left) sweeps $\alpha, \beta \in \{0, 0.25, 0.5, 0.75, 1\}$ on validation; every nonzero point lands in 0.60–0.67 against a base of 0.02, so no mixing ratio interferes with either component. The merge that realizes $\alpha Z_a + \beta Z_b$ is exact (rank-16 vertical and horizontal stacking with the coefficients on the up blocks; tensor-level error at most 10^{-5}), while composing in parameter space after orthogonal alignment deviates from the shift-space composition by roughly 37% relative error and loses over half the marker signal, which is why we state that the correct algebra lives in shift space. The search arms (right) are: text-only mutation and screening over instruction rewrites; coefficient search over a fixed compiled library with nonnegative coefficients capped at 2; and the full loop that mutates text, compiles it, then searches coefficients. All three arms receive exactly T=1300 validation evaluations; the library arm wins because composition makes its search space continuous and reusable rather than discrete and disposable.

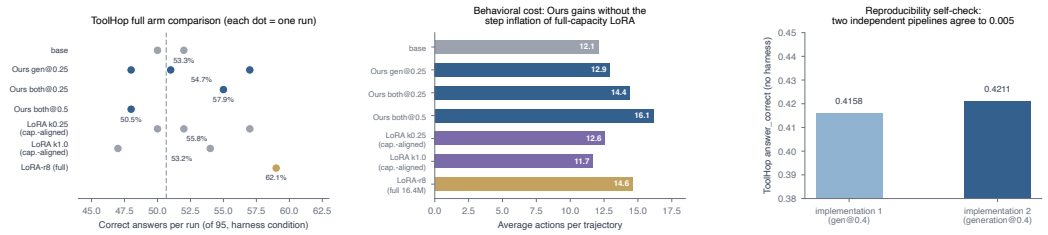


Figure 12: All ToolHop arms with runs as points (left); action counts per arm (middle); reproducibility self-check (right).

G POLICY-COMPILATION DETAILS

Figure 12 (left) shows every ToolHop arm including the full-capacity LoRA reference; each point is one full evaluation, and the spread of single runs, up to $\pm$5–9 items, is why we average repeats and report paired net differences rather than single numbers. The middle panel shows what the gains cost behaviorally: the shift's `both@0.25` gains come with 14.41 average actions against base 11.80, while full-capacity LoRA inflates to 14.61 at over ten times the capacity; the generation-phase shift at the same strength adds only 7%. Gains are not bought with verbosity. The right panel is the reproducibility self-check: two independent implementations of the same configuration differ by 0.005 in the mean, bounding run-to-run noise near half a point on this task, an order of magnitude below the effects we interpret.

H STRENGTH RESPONSE AND FAILURE STAGING

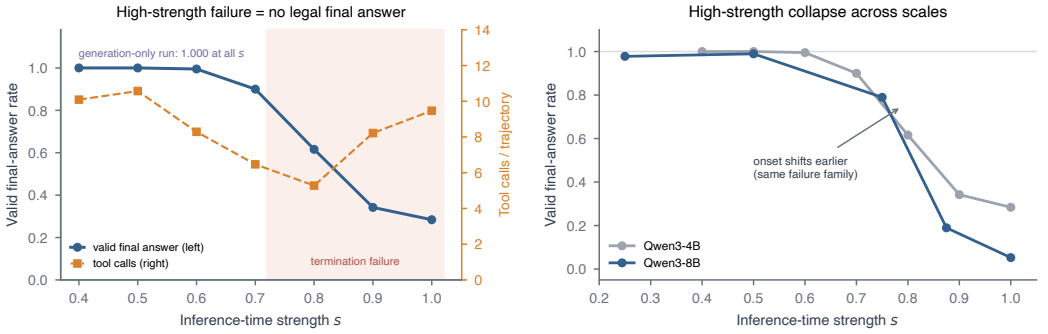


Figure 13: Failure staging at high strength (left); the same failure family at 4B and 8B (right).

The high-strength failure proceeds in two stages (Figure 13, left). Between s=0.7 and 0.8 the policy terminates early: action counts fall from 10.6 to 6.5 to 5.3 while `has_valid_answer` begins to slip. From s=0.9 the behavior degenerates into cycling: action counts rebound to 8–9.5 but valid answers fall to 0.34 and then 0.28. Trajectories always complete and tool-call parsing never fails, so this is a policy-level termination failure, not an artifact, and it is prompt-side specific: generation-only injection holds `has_valid_answer` at 1.000 at every strength tested. The 8B curve (right) shows the same family with an earlier onset (collapse by s=0.875), consistent with the interpretation that the phase boundary is set by the ratio of shift magnitude to the policy's behavioral margin, which shrinks as models get stronger.

I CROSS-SCALE ACCOUNT

The cross-scale protocol moves the same data pipeline, hyperparameters, and evaluation protocol to Qwen3-8B unchanged, in two rounds (our 8B bundle, then the official 8B harness package; the official aligned policy weights were unreleased, so both rounds are harness $\times$ base-policy controlled comparisons). On ToolHop the gain survives at both scales, $+4.0$ to $+4.6$ points with the official

Table 4: Cross-scale account (Qwen3-4B vs. Qwen3-8B): the method skeleton is invariant; gain locations (task × phase × scale) move with the model × task pair.

Dimension	4B	8B	Verdict
ToolHop `both` gain	both@0.25 exactly reproduces 55/95 (+4.3)	multi-band: both@0.5 +4.6 pt, both@0.25 +4.0 pt	gain locatio
SearchQA gain	none (Hotpot subset)	Overall +7.5–9.2 pt (Hotpot +11–15)	direction fli
Phase conclusion	`both` carries the stable gain (≈60% prompt-side)	`gen` ≈ `both` on ToolHop; `both` wins on SearchQA	task × scal
LoRA comparison	same-capacity LoRA ≤ base everywhere	LoRA r11 (3.09× params) matches Ours	Ours match
Scale safe interval	plateau 0.35–0.70; cliff ≥ 0.8	narrows to 0.25–0.5; collapse ≥ 0.875	same failur
API-Bank	zero regression (bit-exact tie)	all arms ≥ base (both@0.25 +4.4 pt)	zero regres

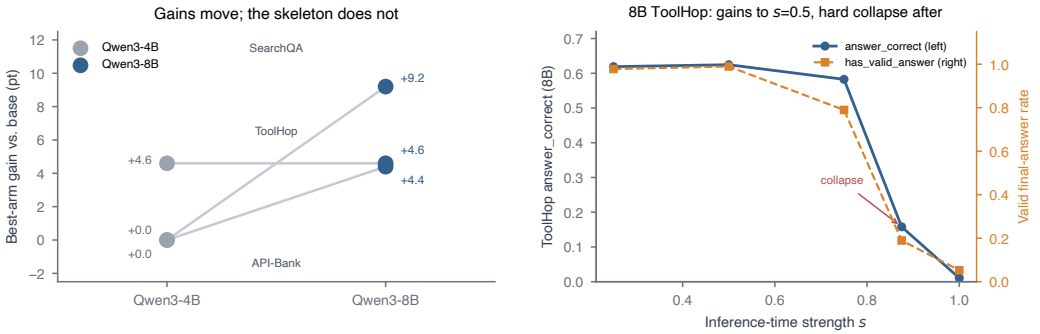


Figure 14: Gain locations across benchmarks and scales (left); 8B ToolHop across strengths (right).

pairing giving +6.3, but it spreads across several strengths instead of concentrating in one configuration. On SearchQA the direction flips: no gain at 4B, +7.5–9.2 points overall at 8B carried by the Hotpot subset (+11–15). On API-Bank the zero-regression property strengthens with scale: every arm is at or above base, with `both@0.25` at +4.4. The LoRA comparisons stay directionally consistent: at 8B, LoRA r11 with 3.09× parameters reaches the same band (62.63% vs. 62.46% on ToolHop; an exact tie at 0.415 on official-protocol SearchQA), never below at equal capacity. What moves across scales is the gain location (task × phase × strength); what holds is the skeleton: zero-weight feasibility, the parameter-efficiency ordering, the failure family, and zero regression.

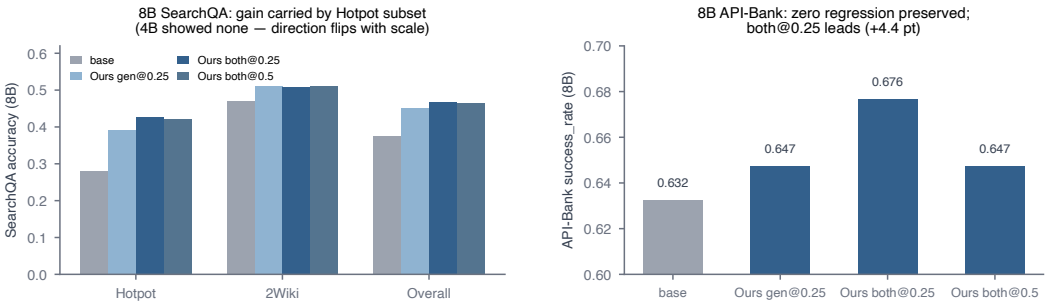


Figure 15: 8B SearchQA by subset (left); 8B API-Bank by arm (right).

Figure 15 splits the two remaining 8B results. SearchQA gains concentrate on Hotpot while 2Wiki moves 2–4 points, within run noise; the official-package round reproduces the Hotpot gain and an overall tie with LoRA r11, so the SearchQA conclusion we draw is subset-specific and scale-dependent. API-Bank at 8B shows the same ordering as at 4B with the signs improved: `api_name_accuracy` is identical across arms (0.9412), meaning the injection does not disturb format selection, and the accuracy edge comes from argument accuracy.

J LLM USAGE DISCLOSURE

Large-language-model tools were used for editing and formatting support during manuscript preparation. All research questions, experimental design, runs, analysis, and claims are the authors' own, and all reported numbers were produced by the experimental pipeline described above.