FROM OPERATION TO INTERFACE: WHERE HARNESS CHANGES ENTER A FIXED MODEL'S TOOL CALLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Procedures that improve a coding agent without touching its weights edit its prompts, tools and middleware and keep the edits that raise task success, but task success alone does not reveal which aspects of tool use an edit changes. We study gpt-oss-20b under six harnesses on SWE-bench Verified and separate each classifiable tool-call attempt into an operation category (such as *search* or *read*) and an invocation interface (shell or API-style). In TOOLS, the layer-20 state at the first generated-token position adds +11.8 accuracy points for the operation but +1.0 for the interface beyond a structured call-history baseline, and an operation decoder transfers across most harnesses and across interfaces. At file-read-matched anchors, TOOLS and SEED differ by +45.8 points in API share, compared with 4.4 points of total variation in the operation distribution, which does not exceed its permutation null. Rewriting past reads between shell and API forms shifts API share by +40.8 and −36.2 points, with smaller operation shifts of 8.6 and 2.5 points, while replacing search hits with no matches shifts the operation by 29.0 points and API share by −6.5. The frozen operation decoder responds most strongly to this result rewrite, but pushing the state along the decoder's direction changes the decoder readout without establishing targeted control of the next operation. These findings separate changes in operation choice from history-sensitive changes in invocation interface.

1 INTRODUCTION

Give a coding agent a new read_file tool and watch its next file read. It may call read_file. It may run sed -n '70,90p' through the shell, as it did before the tool existed. Or it may call open_file, a name the harness never registered, and get an error back. All three happen in the runs we study. Each attempts a file read, yet they differ in interface and in whether the harness can execute them, and a task score cannot tell them apart (Figure 1).

Procedures that improve an agent without touching its weights make exactly this kind of edit, and they keep or discard it by a task score that cannot say where it landed (Lin et al., 2026; Wu et al., 2026; Agrawal et al., 2026; Hu et al., 2025; Zhang et al., 2025; Robeyns et al., 2025). They rewrite the harness, which is everything around the fixed model: the system prompt, the registered tools, what those tools return and any middleware. Concurrent behavioural work compares six tool architectures of comparable capability across three models and finds that what coding agents do changes while their resolve rates stay broadly similar (Xu et al., 2026). We ask where such an edit lands: in what a fixed model is about to do, or only in the interface that carries it.

We locate an edit by splitting each tool call into two halves, the operation it performs and the interface that carries it (Figure 2). Fixed rules give each classifiable tool-call attempt a coarse operation (*search*, *read*, *edit*, *exec* or *list*) and an invocation interface, shell or API-style. Whether the harness registers the name the model writes is recorded separately. We apply them to the calls of one open-weight model, gpt-oss-20b (OpenAI, 2025b), driven by six harnesses on SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024). Of these, three are hand-written, one is a seed harness, and two were built on the seed from the output of improvement procedures.

A call shows the operation the model wrote, but not whether the harness changed what the model was about to do or only led it into a different situation. We therefore compare harnesses at file-read-matched anchors, where both have just read the same file, and read the model's state, its hidden state

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.

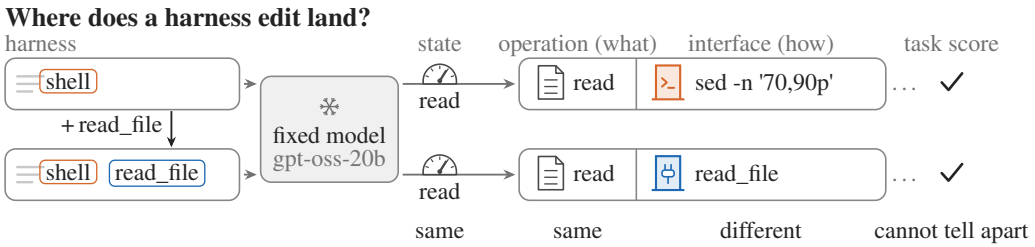


Figure 1: A schematic. An improvement procedure adds tools to the harness, here read_file, and keeps or discards the edit by a task score. The fixed model's next read is the same operation, now possibly written through the new interface, and the score cannot tell the two apart. A readout of the state names the same operation in both rows, but that alone does not settle where the edit landed, so we compare operation distributions, interface use and the readout across harnesses (Section 6).

at layer 20 at the first generated-token position of a call, a median of 48 generated tokens before the call is revealed. Since the context shows every earlier call, we ask what the state adds to the call history, that is, to the operations and interfaces of those calls. Finally, we rewrite one part of the context at a time, or push the state itself, and watch which half moves.

The two halves behave differently (Table 1). The state predicts the coming operation beyond the call history, while the interface tracks mainly the menu and the form of earlier calls. Adding tools to the seed harness moves the API share, the fraction of classifiable calls made in API-style form, far more than the mix of operations, and the resolve rate does not rise significantly (Section 4). At file-read-matched anchors the operation distribution of the next call differs by 4.4 points of total variation, which does not exceed its permutation null, while the API share differs by +45.8 points (Section 6), so an added tool changes mainly the interface. We examine three factors that bear on whether a call takes it: whether the tool can serve the call, the call history, and whether the model writes a registered name (Section 7). We contribute the following.

- **A readout of the coming operation that transfers across most harnesses.** Added to a model of the call history in TOOLS, the layer-20 state raises the accuracy of predicting the operation by +11.8 points, and a decoder fitted on one harness reads the operation on the other harnesses that speak the model's native format and across interfaces (Section 5).
- **Evidence that added tools act on the interface.** At file-read-matched anchors the API share differs by +45.8 points, while the operation distribution differs by 4.4 points, which does not exceed its permutation null. The state adds little incremental predictive value beyond the call history for the interface, and the tested decoder does not establish reliable prediction of whether an attempted API name is registered (Sections 5 to 7).
- **A dissociation of the two halves.** Rewriting past reads between shell and API forms shifts the API share by +40.8 and −36.2 points and the operation by 8.6 and 2.5 points, and rewriting the last search result moves mainly the operation. Pushing the state along the decoder's direction changes the decoder readout without establishing targeted control of the next operation (Section 8).

2 RELATED WORK

Tool interfaces and harness design. How a harness exposes its tools changes what a coding agent does, even when the score stays level. SWE-agent made the agent-computer interface a design variable (Yang et al., 2024), CodeAct lets agents act through executable code (Wang et al., 2024; 2025), and on enterprise tasks an agent given only bash outscores one given typed tools (Mak et al., 2026). Closest to us is the concurrent study of Xu et al. (2026) described in Section 1. Its split of capability from architecture parallels our split of operation from interface, and its finding that typed tools break up compound shell commands has a counterpart in Section 7. Like the studies above, it measures behaviour. We add the model's state, which locates the half of a call that a harness change reached, and rewrites of the context that test the answer one part at a time.

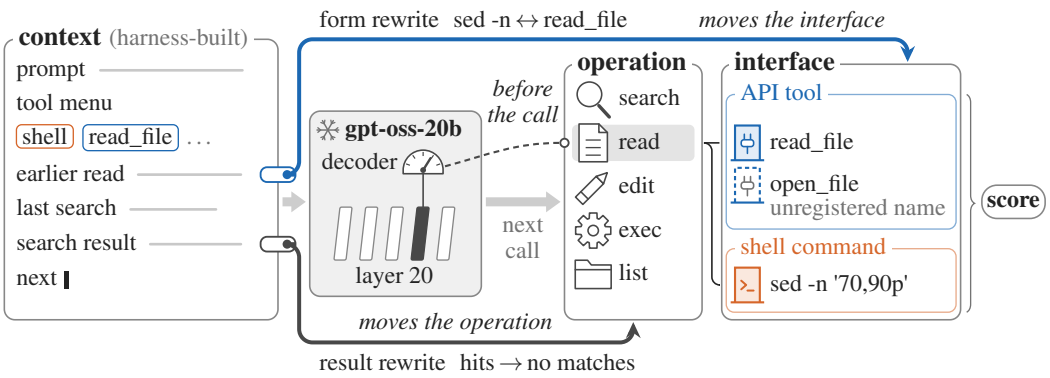


Figure 2: **Harness changes reach the two halves of a tool call in different ways: what a search returns moves mainly the operation, and the form of earlier reads mainly the interface.** A read can be written as read_file, as sed -n, or as open_file, and a task score cannot tell them apart. A decoder reads the operation from the layer-20 state before the call is written. Switches mark the two rewrites of the context, each joined to the half of the call it moves (Table 1).

Tool choice, names and call history. Tool choice is known to depend on names, descriptions and the calls that came before. Small models write tool names that follow pre-training habits rather than the schema, and renaming tools to suit the model reduces such errors (Lee et al., 2026). Names, descriptions and position also bias selection (Blankenstein et al., 2025), and AutoTool exploits tool usage inertia, the tendency of tool calls to follow predictable sequences (Jia & Li, 2026). Within a registered menu, a model can attend to the right tool and still pick a wrong one (Chen, 2026). Our unregistered names are the complementary case, a name outside the menu (Patil et al., 2024; Zhang et al., 2024). We ask what the state carries about names and about the call history (Sections 5 and 7).

Reading an agent's state. A model's state before it acts carries information about what it is about to do. The state predicts whether a tool is needed and how risky the call is, in models that include gpt-oss-20b (Tatsat & Shater, 2026), and the need for a tool is linearly decodable before generation (Sun et al., 2026). Such readouts matter because written reasoning can leave out what drives a choice (Turpin et al., 2023). Lenses show intermediate layers anticipating tokens not yet written (nostalgebraist, 2020; Belrose et al., 2023a; Pal et al., 2023; Lindsey et al., 2025). A decodable property need not be one the model uses (Ravichander et al., 2021; Elazar et al., 2021; Belrose et al., 2023b), so probes need controls (Alain & Bengio, 2017; Hewitt & Liang, 2019) and causal claims usually rest on interventions such as activation addition (Turner et al., 2023; Li et al., 2023; Zou et al., 2023). We test our operation readout against the call history, across harnesses and across interfaces, and intervene on the context by rewriting it and on the state by activation addition. We know of no other comparison of a fixed model's state across real harnesses.

3 Method

Figure 2 gives the method. A harness builds the context of a run, the fixed model writes the next call, and we split that call into its two halves, the operation and the interface. We then locate a harness change in three ways. We compare harnesses at file-read-matched anchors, where both have just read the same file of the same task. We read the layer-20 state before the call is written and ask what it adds to the call history. And we rewrite one part of the context at a time, or push the state itself, and watch which half moves. The rest of this section gives the model, the harnesses, the calls and the readouts.

Model and harnesses. One model is driven by six harnesses that differ in protocol, prompt and the tools they register (Appendix A.1). The model is gpt-oss-20b (OpenAI, 2025b), with 24 layers and a 2880-dimensional residual stream, served by a single vLLM instance (Kwon et al., 2023). Of the three hand-written harnesses (Appendix A.2), HarmonyAgent and Codex (OpenAI, 2025a) speak the model's native Harmony format (Kundel, 2025), and AgentMoE asks for a plain-text

bash block. The seed family also speaks Harmony and starts from SEED, which registers only the shell. EVOLVED is SEED after the improvement procedure AHE (Lin et al., 2026) rewrote its prompt, memory and middleware, without adding an API tool. TOOLS is SEED with the five evolved tools of ModularRSI (Wu et al., 2026), which take a positional args array and cap what they return. The shell stays available in every harness. On SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024), the seed family shares 36 tasks and the hand-written harnesses 50, and resolve rates are scored on the 29 tasks of the seed family admitted for official scoring.

Calls. The analysed calls are replayed from their recorded tokens and assigned an operation and an interface by fixed rules. A proxy records every request and response, and replay feeds the recorded tokens back under teacher forcing, so each state is one the model computed for the call. Replay is token-exact for HARMONYAGENT and the seed family and close to exact for AGENTMOE and CODEX (Appendix B.1). The operation classifier assigns each classifiable tool-call attempt one of five coarse operations, *search*, *read*, *edit*, *exec* or *list*, and the interface records whether the attempt took API-style or shell form. Registration validity is recorded separately: unregistered API-style attempts keep both labels in the merged-set analyses and are reported separately in the adoption and rewrite analyses. A search is grep_search or a shell grep, and a read is read_file or sed -n 'A,Bp'. In all, 8 172 calls are classifiable. Heredoc scripts that print file contents are ambiguous and count as *exec*, except in the adoption counts of Section 7, where they count as *read* (Appendix B.2).

State and readouts. We read the layer-20 state at the first generated-token position of each call, where the model generates the first token of the turn that issues the call (a channel marker in the Harmony harnesses, the first reply token in AGENTMOE). The call is revealed a median of 48 generated tokens later. The axis comparisons of Section 4 instead average the state over the written tokens of whole runs (Appendix B.3). The frame is the released 64-axis J64 frame for this model (Chen et al., 2026), a fixed linear basis of the layer-20 residual stream that we never refit. Its release designates 16 named axes, which we test as one family with Holm's correction (Holm, 1979). A decoder is a multinomial logistic regression on the 64 frame coordinates, cross-validated over five folds grouped by task. We score a single decoder by its normalised gain, $(\mathrm{acc} - \mathrm{base})/(1 - \mathrm{base})$ in points, where base is the majority-class rate. A frozen decoder, fitted once on the recorded calls of all six harnesses, is applied unchanged in Section 8, to rewritten contexts and to states pushed along its own direction. Paired comparisons are within task or within context, by sign tests with ties excluded or against permutation nulls. A second readout, the agent lens, is a Jacobian lens (Gurnee et al., 2026) built from replays of the seed family (Appendix C.2).

What the frame contributes. The frame is a fixed measurement basis, not the source of what we decode: a random 64-dimensional projection decodes the operation about as well (normalised gain 25.2 against 25.4 points on the baseline set of Appendix C.3). What the frame adds is named axes and a shared SD scale, which Sections 4 and 6 use to compare harnesses axis by axis.

4 RQ1: WHAT DOES A HARNESS CHANGE?

The hand-written harnesses differ in everything we measure, but within the seed family, adding tools moves the interface and little else (Table 1, rows 1–4). We measure the operation shift as the total variation distance (TV) between operation mixes and the interface shift as the change in API share, both in points. The named axes, averaged over each run, and the operation mix differ most in pairs with AGENTMOE, which writes plain-text commands and edits more than any other harness. Between TOOLS and SEED (row 3), 0 of the 16 named axes pass Holm, and the operation shift of 6.5 points and the decoder TV are close to those of EVOLVED, which rewrote its prompt, memory and middleware (row 4). The API share instead rises from none in SEED to 43.5% in TOOLS.

Neither these averages nor the resolve rate says whether TOOLS changed what the model does or only how it does it. TOOLS resolves 10 of the 29 admitted instances and SEED resolves 4, a difference that is not significant (exact McNemar $p = 0.11$). The small operation shift does not settle it either, because each harness leads the model into situations of its own and an average over them can hide changes that cancel. Telling these apart needs a readout of the coming operation that can be compared across harnesses (Section 5) and a comparison at file-read-matched anchors (Section 6).

Table 1: **Adding tools or rewriting the form of earlier reads moves mainly the interface, rewriting the last search result moves mainly the operation, and a push along the decoder's direction moves the readout without establishing targeted control of the next operation.** Shifts in points on the same classifiable calls: *Operation*, TV over five operations; *Interface*, change in API share; null_{95}, the 95th percentile of the permutation null. *State*: decoder TV and named axes (of 16) passing Holm. Brackets: 95% bootstrap CIs; bold: beyond the null ($p < 0.05$).

		Operation		Interface		State	
Comparison	n	TV	null_{95}	Δ API share	null_{95}	Decoder	Axes
Harnesses, whole runs (n = tasks)[a]							
1 CODEX vs HARMONYAGENT	50	8.7 [5.0, 14.4]		−52.6 [−59.6, −45.2]		9.0	5
2 AGENTMOE vs HARMONYAGENT	50	15.9 [10.0, 22.9]		−58.5 [−65.2, −51.3]		15.5	10
3 TOOLS vs SEED	36	6.5 [3.2, 11.4]		+43.5 [+32.8, +54.4]		2.8	0
4 EVOLVED vs SEED	36	5.8 [3.1, 11.5]		+0.1 [0.0, +0.2]		3.1	0
File-read-matched anchors, the next call after both harnesses have read the same file (n = tasks)[b]							
5 Anchor, TOOLS vs SEED	32	4.4 [2.4, 12.5]	9.7	**+45.8** [+32.7, +58.5]	20.9	4.9	3 · 1
6 Anchor, TOOLS vs EVOLVED	32	3.4 [2.1, 13.2]	10.5	**+45.1** [+32.3, +58.5]	20.3	5.5	2 · 3
7 Strict anchor[c], TOOLS vs SEED	15	10.1 [3.4, 26.7]	17.6	**+11.6** [+3.7, +22.1]	9.5	4.0	0 · 0
8 Strict anchor, TOOLS vs EVOLVED	15	16.9 [7.9, 33.2]	19.2	**+11.6** [+2.8, +24.4]	10.7	5.3	0 · 0
9 Anchor[d], EVOLVED vs SEED	32	4.3 [2.0, 12.3]	8.2	0.0 [0.0, 0.0]	0.0	1.3	0 · 0
Rewrites of one part of the context, the rest byte-identical (n = contexts)[e]							
10 Form: shell → API	60	**8.6** [3.2, 16.3]	8.4	**+40.8** [+30.4, +51.2]	15.5	**2.2**	0
11 Form: API → shell	60	2.5 [2.1, 9.8]	7.8	**−36.2** [−46.7, −25.9]	13.9	0.8	0
12 Result: hits → no matches	90	**29.0** [20.7, 38.0]	12.0	**−6.5** [−12.7, −0.9]	6.1	**18.1**	8
13 Result: no matches → hits	19	12.1 [6.6, 22.0]	13.3	−7.8 [−26.4, +8.3]	15.9	**7.6**	2
Pushes of the state along the decoder's direction, against no push (n = contexts)[f]							
14 Toward *read*, $\alpha = -2$	40	7.6 [2.9, 17.5]	10.9	−3.7 [−10.4, +3.8]	6.8	**47.4**	—
15 Toward *read*, $\alpha = -4$	40	6.9 [2.2, 18.0]	12.1	+0.8 [−7.0, +8.7]	7.7	**55.3**	—
16 Toward *search*, $\alpha = +2$	40	9.2 [3.5, 18.1]	10.7	−0.4 [−6.4, +5.3]	5.9	**48.2**	—
17 Toward *search*, $\alpha = +4$[†]	40	12.1 [5.9, 33.2]	23.7	**+49.6** [+24.4, +68.7]	19.6	**60.0**	—

[a] Shared tasks, without one-to-one matching of calls. These comparisons are descriptive, with no permutation null and no marks (Table 11 gives a re-split null and every within-group pair). Decoder fitted on these calls, axes averaged over whole runs (Appendix D.1). Unregistered API-style names count as API-style (row 4). [b] Decoder refitted with whole tasks held out. Axes at call $t \cdot t{+}1$. Unregistered next calls in neither half. [c] Call t via the shell in both. [d] Negative control: no API tool in either, so every task ties. [e] Rewritten minus original. Frozen decoder of rows 1–4. Unregistered names in neither half. 6 (form) and 8 (result) continuations per version of at most 256 tokens. [f] Original contexts of row 12, 3 continuations per context and condition, with α token-level SDs of the decoder's direction from *read* to *search* added to the state at 24 positions from the last prompt token on (Appendix H.4). Decoder: how far the push moved it, a check that the push took hold. [†] Under 50 classifiable calls in a version: the push breaks the output and leaves 21 calls, mostly API calls, so the bold interface shift reflects which calls survive.

5 RQ2: WHAT DOES THE STATE ADD TO THE CALL HISTORY?

Both halves of the coming call can be decoded from the state, but that alone says little, because at the first generated-token position the model has already read every earlier call. On the 1 334 calls of TOOLS that follow an earlier call, a decoder on the frame reaches a normalised gain of 30.6 points for the operation and 36.0 for the interface. We therefore set the state against a model of the call history rebuilt from the full trace. Its features are the operation and the interface of the previous call and their interaction, the API share over the last three calls and over all earlier calls, and the position of the call in its trajectory. The question is what the state adds to that history, and here the two halves differ.

Beyond the call history, the state raises the accuracy of the operation prediction by +11.8 points but barely raises that of the interface, by +1.0 (Figure 3a). This increment over the call history is the accuracy of a model given both the history and the state, minus that of the history alone. For the interface the history alone already beats the state alone, while for the operation the state alone beats the history. Each increment is compared with 1 000 refits in which the states are shuffled among the calls of the same task, which keeps what the state carries about the task and removes what it carries about the particular call. The operation increment exceeds every refit, and the interface increment

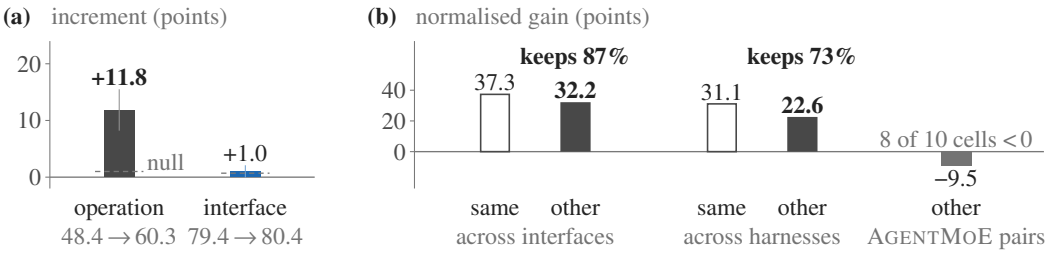


Figure 3: **The operation readout beyond the call history, and its transfer.** (a) Gain in accuracy from adding the layer-20 state to a model of the call history, for the operation and for the interface, on 1 334 TOOLS calls (whiskers: 95% task-bootstrap CI; dashes: permutation null). (b) Normalised gain of the operation decoder trained on the same interface or harness (white) and on another one (dark); grey: mean over the 10 AGENTMOE cells.

exceeds nearly all of them (p = 0.009). At the first generated-token position, then, the state carries information about the coming operation beyond this structured call history, while what it carries about the interface is almost entirely in those calls already.

The asymmetry holds under every readout we tried and in every harness (Appendix E.1). Across the frame, the agent lens and random projections of the raw state, the operation gains +2.3 to +14.6 points in all six harnesses and the interface −0.7 to +1.0 in those whose calls use both interfaces (Appendix E.1). Even within searches alone or reads alone, where the operation is fixed, the state adds nothing detectable to what the history predicts about the interface (Appendix E.2). An operation readout is useful for comparing harnesses only if it transfers between them.

A decoder fitted on one harness reads the operation on the others and keeps most of its normalised gain (Figure 3b). Among the harnesses that speak Harmony, all 20 transfer cells exceed a null of decoders fitted on shuffled labels (Appendix E.4). Their mean gain is 73% of that of decoders cross-validated on the target harnesses. It still works across harnesses that ran mostly different tasks, if less well than across those that shared them, so it does not rest mainly on memorised tasks. AGENTMOE, the only harness that writes plain-text commands, is the exception, with only 3 of its 10 cells above the null. Part of this gap comes from its mix of operations. AGENTMOE edits far more often than the others, and once the class counts are balanced, all its cells rise above zero but stay below every other pair (Figure 6 in Appendix E.5). Transfer across harnesses could still depend on how a harness writes its calls, so we test transfer across interfaces inside one harness.

Within one harness, a decoder trained on calls made through one interface reads the operation of calls made through the other (Figure 3b). TOOLS makes many reads and searches through both interfaces, so we can hold the harness, the prompt and the tasks fixed and vary only the interface. We balance the two operations and keep every task on one side of the split. Across interfaces the decoder keeps 87% of its normalised gain, a mean of 32.2 points against 37.3 within one interface. With all four operations the two interfaces share, the classes shrink and the transfer weakens (Appendix E.6). This transfer lets us compare harnesses through the readout at file-read-matched anchors (Section 6).

6 RQ3: AFTER A SHARED FILE READ, WHAT DIFFERS?

To separate what a harness changes from the situations it creates, we compare TOOLS and SEED at file-read-matched anchors, where both have just read the same file of the same task. The two trajectories developed separately up to that read, so an anchor matches the file, not the call history or a shared snapshot. The shared read is call t, and we compare the next call, t+1, within each of 32 tasks. SEED registers no API tool, so the comparison measures how often TOOLS uses its tools where SEED can only use the shell.

At the next call the interfaces differ, while the operation distribution does not exceed its permutation null (Table 1, rows 5–9). The operation shift is 4.4 points of total variation (p = 0.61), while on the same classifiable calls the API share differs by +45.8 points and is higher for TOOLS in every non-tied task. Even the upper end of the operation's 95% interval lies below the lower end of the in-

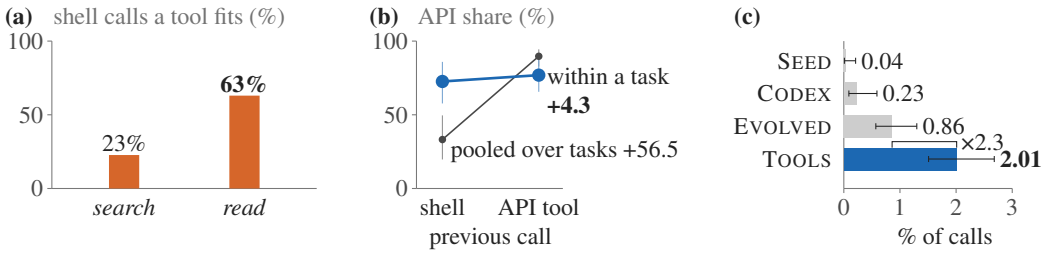


Figure 4: **Three factors in whether an offered API tool is used.** (a) Share of TOOLS' shell searches and shell reads that one API call could have served. (b) API share of an applicable call after an API call and after a shell call, pooled over tasks and averaged within tasks (whiskers: 95% task-bootstrap CI). (c) Share of calls to an unregistered tool name, per harness (Wilson 95% CI).

terface's. In the negative control, EVOLVED against SEED, the interfaces tie on every task and the operation shift again stays inside its null. The search anchor and every treatment of heredoc scripts give the same pattern (Appendix F). What the model was about to do could still differ, so we read the state.

Comparing states needs a tighter match than the anchor gives. In a balance check at call t, 3 and 2 of the 16 named axes already separate TOOLS from SEED and from EVOLVED (Table 1, rows 5 and 6). At many of these anchors TOOLS read the file through read_file, which SEED cannot call, so the anchor matched the file but not the call. The strict anchor therefore also requires call t to go through the shell in both. On its 15 tasks, no named axis separates either pair at call t.

On the strict anchor the readouts of the next call do not separate the harnesses, while the interface of that call still differs (Table 1, rows 7 and 8). The decoder's operation distributions for TOOLS and SEED differ by 4.0 points ($p = 0.49$) and no named axis separates them, and their API shares differ by +11.6 points. Its behavioural operation shifts, 10.1 and 16.9 points, are as large as the interface shift and stay inside their nulls, so on 15 tasks it cannot resolve operation differences of that size. Appendix F.5 reads the decoder on the search anchor, where against EVOLVED it exceeds its null.

The interface difference is smaller on the strict anchor, and it tracks how TOOLS made the shared read. Averaged over tasks, the next call of TOOLS uses an API tool 64.8% of the time after a read through read_file and 14.0% after a shell read (Appendix F.2). Within the tasks that contain both kinds of anchor the contrast is weaker, so part of it is a preference of the task (Section 7).

7 RQ4: WHEN IS AN OFFERED TOOL USED?

We examine tool applicability, task-stratified history dependence and naming validity (Figure 4).

Applicability. An API tool fits most shell reads, not most shell searches (Figure 4a), so applicability accounts for much of the apparent non-adoption of searches but little of that of reads. A shell call is applicable when one API call would return the same content. Most shell searches of TOOLS are compound commands that grep_search cannot express, like those that typed tools break apart in Xu et al. (2026). The API share of searches rises from 38.9% over all searches to 73.7% over applicable ones. Most shell reads, in contrast, are sed -n line ranges that read_file could have returned, and over applicable reads the API share is only 56.4%.

Task and call history. On applicable calls, the pooled association between consecutive interfaces weakens substantially after task stratification (Figure 4b). After an API call, TOOLS uses an API tool on 89.7% of applicable calls, and after a shell call on 33.2%. Stratified by task, the odds ratio is 1.45 [0.84, 2.51] ($p = 0.15$), while over all consecutive calls a within-task association remains, with an odds ratio of 2.43 [2.05, 2.89] (Appendix G.2). This concerns the immediately preceding call, and rewriting the form of several earlier calls still changes interface use within one context (Section 8). Beyond this history the state adds little incremental predictive value for the interface (Section 5), and the model-based annotation labels none of the 712 applicable calls as stating an explicit interface rationale (Appendix G.3).

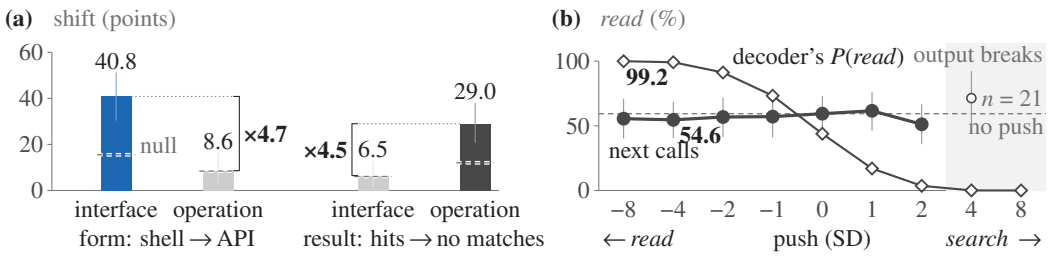


Figure 5: **Rewriting the context and pushing the state.** (a) The forward rewrites, the reverse ones being in Table 1: absolute change in API share (interface) and total variation (operation), in points on the same classifiable calls (whiskers: 95% context-bootstrap CI; dashes: permutation null). (b) The decoder's $P(\text{read})$ and the *read* share of the next calls as the state is pushed along the decoder's direction, over 40 contexts (shaded: pushes that break the output).

Unregistered names. The third factor is the name itself: the model reaches for an evolved tool by a name the harness never registered. On the trajectory set (Appendix A.4), TOOLS does so on 2.01% of its calls, more often than SEED, EVOLVED and CODEX, the other harnesses whose requests carry a menu (Figure 4c). The names are near misses: 95.6% are closest to one of its evolved tools, most often search_file for grep_search. The arguments stray from the menu as well. Of the 60 such calls we checked (Appendix G.4), 42 use named keys, like the tools of HARMONYAGENT, while every call to a registered evolved tool uses the menu's positional args array.

Names in the state. At this layer and position, the tested decoder does not establish reliable prediction of whether an attempted API name is registered. Among replayed calls that reach for an evolved tool, it tells unregistered names from registered ones at AUC = 0.592, against a null of 0.504 ± 0.060 (Appendix G.5). The harness could instead register the names the model already produces (Lee et al., 2026) and accept named arguments (Appendix H.5).

8 RQ5: CAN THE TWO HALVES BE MOVED SEPARATELY?

If the operation and the interface are set by different parts of the context, rewriting one part should move one half and leave the other. We take real TOOLS contexts, rewrite one part of each and leave the rest byte-identical (Figure 2). From the original and the rewritten version of each context we sample continuations and compare the calls they issue.

Each rewrite is aimed at one half. The form rewrite changes how earlier file reads in the call history were written. Every shell range read, sed -n 'A,Bp' f, becomes the equivalent read_file call, or the reverse, and what each call returned stays byte-identical. We expected it to move the interface and not the operation. The result rewrite leaves every call alone and changes what the last search in the context returned. A hit list becomes {"content": "no matches"}, the reply grep_search gives when nothing matches, or such a reply becomes a hit list from another search on the same task. We expected it to move the operation, since a search that found nothing leaves no line to open, and to leave the interface roughly in place, since the menu does not change.

Rewriting the form of earlier reads moves the interface in both directions and the operation far less (rows 10–11 of Table 1, and Figure 5a). Turning shell reads into read_file calls moves the API share of next calls by +40.8 points, and the reverse rewrite moves it by −36.2, in nearly every context whose API share changes (Appendix H.1). The observed ratios of the interface shift to the operation shift are 4.7× and 14.3× in the two directions. From shell to API the operation shift only just clears its permutation null, as some searches give way to reads, while from API to shell it stays well inside it. The no-tool-call rate remains similar, but registration errors change substantially. Continuations that call an unregistered name rise from 0 to 34 under the shell-to-API rewrite and fall from 54 to 15 in the reverse direction (Appendix H.1). Two earlier runs on fewer contexts move the API share the same way, and in neither does the operation shift exceed its permutation null (Appendix H.3).

Rewriting what the last search returned moves mainly the operation (rows 12–13 of Table 1, and Figure 5a). When a hit list becomes no matches, the operation shifts by 29.0 points, mostly as reads

give way to searches. The API share moves as well, beyond its null, but the operation shift is 4.5 times as large. The reverse rewrite moves neither half beyond its null (row 13).

The frozen decoder of Section 3 moves most where the operation moves most, its four shifts in the order of the four operation shifts (rows 10–13 of Table 1). When a hit list becomes no matches, its mean posterior shifts by 18.1 points and its $P(\text{read})$ falls with the *read* share. Under the form rewrite it shifts by at most 2.2 points, just beyond its null in one direction, though the API share moves by at least 36.2 points. From no matches back to hits it just clears its null while the operation does not (row 13).

The decoder predicts the coming operation from the state, but pushing the state along its direction changes the decoder readout without establishing targeted control of the next operation (rows 14–17 of Table 1 and Figure 5b). On contexts whose last search returned hits, we added the decoder's direction from *read* to *search* to the layer-20 state, with either sign, and sampled continuations (Appendix H.4). Pushed toward *read* by four token-level SDs, the decoder's mean $P(\text{read})$ reaches 99.2% (row 15), while 96% of continuations remain well formed, compared with 86% without a push, and the observed *read* share does not increase. Increasing the push toward *search* degrades output validity without establishing a targeted increase in search calls. At four SDs only 18% of continuations issue a classifiable call, most of them API calls (row 17). A random push of the same norm barely moves the decoder, and no push along the direction differs from it in the share of the operation it points to, at any size of push (Fisher's exact test, $p \geq 0.53$).

9 Discussion

The harness changes we studied reach a fixed model's tool calls in two ways: what the tools return reaches mainly the operation, and the menu and the form of the model's own past calls reach mainly the interface. The state predicts the operation beyond the call history and adds little for the interface, whose use tracks the menu, the registered names and the form of past calls (Sections 5 to 8). The evolved tools of TOOLS enter the second way, through the interface.

For procedures that evolve harnesses, an added tool can be checked as an interface change before it is scored. Its adoption should be counted on applicable calls, and since calls in the context pull the next call toward their form, the harness can show the tool in use early. In our form rewrite this also drew calls to unregistered names (Appendix H.1), so the names the model reaches for should be registered with its argument convention. With open weights, the operation readout adds one more check beside the operations and interfaces of the sampled calls and whether their names are registered. We give, but did not run, a selection protocol built on these checks (Appendix H.5).

Limitations. Everything here comes from one model read at one layer, on one benchmark and one set of added tools, and interface effects vary with the model (Xu et al., 2026). The readouts concern the next call, and 29 scored instances are too few to link either half to task outcomes. The file-read-matched anchors pair trajectories that developed separately rather than from one shared snapshot, and the strict anchor has too few tasks to resolve small differences in the axes or in the operation. Our rewrites never changed the menu and were never crossed in the same contexts. Our pushes of the state, each along one direction at one layer, did not establish targeted control of the next call. What the state adds was measured against the structured call history, not the full transcript, and at the first generated-token position only, so an interface or a name settled later would not show.

10 Conclusion

The tools we added to a fixed model's harness changed mainly the interface of its calls, as did rewriting the form of earlier reads, while rewriting the last search result moved mainly the operation. The layer-20 state predicts the coming operation beyond the call history, but pushing it along the decoder's direction does not establish targeted control of the next operation. Splitting each call into its two halves gives any harness edit a question to answer before it is scored: which half did it move?

ETHICS STATEMENT

This work studies a publicly released open-weight model on a public software-engineering benchmark and involves no human subjects and no personal data. Agents ran directly on repository checkouts from a local bare cache, without container isolation, and the CODEX harness ran with its sandbox set to full access. Only scoring used the official per-instance images, entered through an unprivileged user-namespace container. Running agents without isolation let one of them write outside its task. In an early run, a harness executed its tools inside the launcher process, whose working directory was our own source tree rather than the task checkout, and the agent's calls left files there, including a sitecustomize.py. We discarded that run, moved the files out of the tree and changed the launcher so that it refuses to start unless the shell points at the task checkout. No reported result comes from that run. The recording proxy stores model requests and responses for replay and nothing else. The additive pushes of Appendix H.4 were measurements, not a steering method.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A.1 give the model, the serving stack, the benchmark and the six harnesses, Appendix A.2 lists the changes we made to each harness, and Appendix B.1 describes token recovery and replay validation. The J64 frame is the released one (Chen et al., 2026) and is never refitted. The agent lens is not a released artefact. We built it from replays of the seed family, the same harnesses whose calls are decoded in Section 5, with 16 reference states and 14 concept families (Appendix C.2). The operation classifier and its command extractor are given in Appendix B.2, the applicability rule in Section 7 and Appendix G.1, and the paired tests, permutation nulls and the Holm family of 16 named axes in Section 3. The two context rewrites and the frozen decoder read on them are specified in Section 8 and Appendix H and the pushes in the state in Appendix H.4. The analysis scripts and the number registry, which maps every number in the paper to the output file and key it comes from, are included in the anonymised supplementary material. Recorded traces and replayed states will be released with the paper.

AI USE STATEMENT

AI assistants were used in three ways. An AI-assisted audit helped refine the rule-based classifier described in Appendix B.2: a language model judged a sample of shell commands blind to the rule-based label, and its judgements replace no label and enter no reported number. Separately, gpt-oss-20b annotated the reasoning preceding 712 applicable calls, and these annotations are used in Section 7 and Appendix G.3. AI assistants also helped look up and check related work. The authors take full responsibility for the content.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2017. arXiv:1610.01644.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens, 2023a. arXiv:2303.08112.

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. LEACE: Perfect linear concept erasure in closed form. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b. arXiv:2306.03819.

Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. BiasBusters: Uncovering and mitigating tool selection bias in large language models, 2025. arXiv:2510.00307.

Kang Chen, Sihan Zhao, Yixin Cao, and Yu-Gang Jiang. Beyond the trace: Coupling an interpretable reasoning-state readout to native MoE routing, 2026. arXiv:2608.17638.

Shiyang Chen. Looking is not picking: An attention-segment account of tool-selection failures in LLM agents, 2026. arXiv:2606.16364.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI, https://openai.com/index/introducing-swe-bench-verified/, 2024.

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021.

Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models, 2026. arXiv:2607.15495.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations (ICLR)*, 2025.

Jingyi Jia and Qinbin Li. AutoTool: Efficient tool selection for large language model agents. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2026. arXiv:2511.14650.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Dominik Kundel. OpenAI Harmony Response Format, August 2025. URL https://developers.openai.com/cookbook/articles/openai-harmony. OpenAI Cookbook.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.

Jonggeun Lee, Woojung Song, Jongwook Han, Haesung Pyun, and Yohan Jo. Don't adapt small language models for tools; adapt tool schemas to the models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. arXiv:2510.07248.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. Transformer Circuits Thread, https://transformer-circuits.pub/2025/attribution-graphs/biology.html, 2025.

Hazel Mak, Susheel Suresh, Sahil Bhatnagar, Barry Wang, Chhaya Methani, and Alejandro Gutierrez Munoz. Is Bash all you need? an empirical study of tool interfaces for enterprise digital worker agents, 2026. arXiv:2609.11999.

nostalgebraist. Interpreting GPT: the logit lens. LessWrong, https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens, 2020.

OpenAI. Codex CLI: Lightweight coding agent that runs in your terminal. https://github.com/openai/codex, 2025a.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card, August 2025b. URL https://cdn.openai.com/pdf/419b6906-9da6-406c-a19d-1bb078ac7637/oai_gpt-oss_model_card.pdf.

Koyena Pal, Jiuding Sun, Andrew Yuan, Byron C. Wallace, and David Bau. Future lens: Anticipating subsequent tokens from a single hidden state, 2023. arXiv:2311.04897.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2305.15334.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. Probing the probing paradigm: Does probing accuracy entail task relevance? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 3363–3377, 2021.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent, 2025. arXiv:2504.15228.

Chung-En Sun, Linbo Liu, Ge Yan, Zimo Wang, and Tsui-Wei Weng. LLM agents already know when to call tools – even without reasoning, 2026. arXiv:2605.09252.

Hariom Tatsat and Ariye Shater. Beyond the black box: Interpretability of agentic AI tool use, 2026. arXiv:2605.06890.

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

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.04388.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2402.01030.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. ModularRSI: Modular and generalizable recursive harness self-improvement, 2026. arXiv:2609.14857.

Xiangzhe Xu, Hamidreza Saghir, Qianhui Wu, Marc-Alexandre Côté, Tong Wang, Kiran Lakkaraju, Kexin Pei, and Xiangyu Zhang. The devil is in the interface: Evaluating how tool architecture shapes coding agent behavior. In *Conference on Language Modeling (COLM)*, 2026. arXiv:2608.11386.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents, 2025. arXiv:2505.22954.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. ToolBeHonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 11388–11422, 2024.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to AI transparency, 2023. arXiv:2310.01405.

A THE HARNESSES

The six harnesses share the model and the server and differ in protocol, instructions, menu and client settings.

A.1 WHAT EACH HARNESS REGISTERS

Table 2 lists the protocol and the menu of each harness. HARMONYAGENT and CODEX speak Harmony, AGENTMOE asks for a plain-text THOUGHT: block followed by a fenced bash command, and the seed family speaks Harmony through the Responses API. The shell is reachable in every harness. Within the seed family the instructions differ mainly in length: the median rendered instructions of a request are 586 bytes for SEED, 9 188 bytes for EVOLVED and 589 bytes for TOOLS. EVOLVED also carries long-term memories and a middleware (Appendix A.3). TOOLS keeps the instructions of SEED and adds the five evolved tools, which take a positional args array and return bounded content.

Each harness samples with the settings of its own client, and the proxy recorded them with every request. All 10 180 requests of the seed family ask for high reasoning effort, a temperature of 0.7, a top-p of 0.95 and at most 32 000 output tokens, so these settings cannot separate the three. HARMONYAGENT and AGENTMOE sample at a temperature of 1.0 with a top-p of 1.0. HARMONYAGENT states high reasoning effort in its prompt and caps each completion at 4 096 tokens, while AGENTMOE sends no reasoning setting. CODEX sends no temperature, so the server default applies, and asks for high reasoning effort in 2 098 of its 2 115 requests and for low effort in the rest. Between the hand-written harnesses these settings are therefore part of what differs.

A.2 ADAPTATIONS RELATIVE TO THE STOCK HARNESSES

Running the hand-written harnesses against one local server required changes. Serialisation faults between the server and the stock clients were repaired in the proxy and leave what the model sees unchanged.

The remaining changes alter what the model sees. The web_search and multi-agent tools were removed because the server rejects them. Stock CODEX declares apply_patch as a grammar-constrained custom tool, which the server does not support, so we declared it as a function tool. The model therefore writes patches as escaped JSON strings without constrained decoding.

Table 2: **The harnesses.** Classified calls are the calls of the merged set to which the operation classifier assigns an operation (Appendix A.4), which sum to 8 172.

Harness	Protocol	Tasks	Calls	Menu
HARMONYAGENT	Harmony, completions	50	910	shell (container.exec, an argv array); repo_browser.{search, open_file, apply_patch, print_tree, list_dir} with named parameters
CODEX	Harmony, Responses	50	2 023	exec_command (a cmd string run through bash -lc); apply_patch
AGENTMOE	plain text	50	737	fenced bash only
SEED	Harmony, Responses	36	1 534	run_shell_command only
EVOLVED	Harmony, Responses	36	1 604	run_shell_command only; memories and middleware
TOOLS	Harmony, Responses	36	1 364	run_shell_command; read_file, grep_search, glob_files, edit_file, write_file with positional args

Agents ran directly on repository checkouts and were scored in the official environment. The per-instance SWE-bench images were not available when the agents ran, so repositories were checked out from a local bare cache without container isolation, and CODEX ran in its full-access sandbox mode. The checkouts contain the tests, which no harness could run without the per-instance environment. The resolve rate of Section 4 was scored with the official per-instance images, pulled as layers and entered through an unprivileged user-namespace container. An instance was admitted for scoring only if, in its official image, the FAIL_TO_PASS tests fail with the test patch alone and pass with the gold patch. The admitted set was fixed before any harness was scored and holds 29 of the 36 tasks.

A.3 THE MIDDLEWARE OF EVOLVED

EVOLVED differs from SEED in more than its prompt, because a middleware appends notes to tool outputs, and in the trajectories these notes look like part of what the tools returned. The middleware watches the trajectory and, when a risk pattern repeats, appends a short note to the end of the tool output. Notes that the model has not yet seen are kept as reminders for the next call. Among the rules that fired in our runs are "the same class of error has just repeated; stop retrying the same command pattern, check the contract, path or tool signature once, then change strategy" and "this is only a help, syntax or file-existence check and is not final proof; run the evaluator-style real entry point". The long-term memories concern the same discipline of verification.

A.4 WHICH SUBSET EACH NUMBER COMES FROM

Recording, replay, classification and the naming analyses admit different sets of calls, so the counts in this paper are taken over several subsets of the same traces (Table 3). Each caption in the paper names the subset it uses.

Two pairs of counts differ by subset and rule. The first is the API counts of Table 4 and Table 19. The operation classifier maps an unregistered name into the operation that the name implies, so Table 4 counts attempts, while the adoption analysis counts calls to registered tools. For search the 229 attempts are the 185 calls to grep_search plus 27 calls to search_file, 14 to search and 3 to search_files. For read the 273 attempts are the 260 calls to read_file plus 13 calls to open_file. The second pair is the shell reads, 226 in Table 4 against 319 in Table 19. The stored labels treat the heredoc family as *exec*, while the adoption counts treat it as *read* (Appendix B.2).

The counts for TOOLS differ by one filtering step at a time. Of the 1 370 replayed calls that carry a state, 1 364 receive an operation and enter every decoder, and the other 6 fit no rule. The history control of Appendix E.1 uses the 1 334 of them that follow at least one earlier tool call.

Table 3: **Data subsets.** Every count in the paper is taken on one of these.

Subset	Admission rule	Size
Trajectory set	Recorded calls, deduplicated by call id, from the longest request of each task whose instance id a recorded request names	2 702, 2 562, 2 239 and 1 743 calls on 27, 29, 25 and 47 tasks in SEED, EVOLVED, TOOLS and CODEX
Replay batches	The seed family was replayed in a batch of 24 tasks with at most 40 calls per task and an earlier batch of twelve tasks	36 tasks
Merged classified set	Replayed calls with a state at the first generated-token position to which the classifier assigns an operation	2 023, 1 604, 1 534, 1 364, 910 and 737 in CODEX, EVOLVED, SEED, TOOLS, HARMONYAGENT and AGENTMOE, 8 172 in all
History-control set	Calls of the merged set that follow at least one earlier tool call	1 334 in TOOLS, 903 in HARMONYAGENT
Baseline set	Replayed seed-family calls labelled in a separate pass under the read treatment, with unregistered names left out and unassigned shell commands kept as a sixth class. All readouts of Table 7 are fitted on it	4 604, 168 of them in the sixth class
Case-study set	Baseline-set calls whose action is revealed after the eight-token readout window	4 542
Anchor	Same task, both harnesses have just read the same file, every next call kept	32 tasks
Stored-state and strict anchor	Anchor calls with a stored state. The strict anchor also requires call t through the shell in both harnesses	32 to 34 tasks, 15 on the strict anchor
Naming, trajectory pass	Unregistered names on the trajectory set	45 of 2 239 in TOOLS
Naming, replay pass	Unregistered names among replayed calls	59 of 1 370 in TOOLS
Naming, argument pass	Unregistered calls read for their parameter keys	60 of 1 371 in TOOLS
Reasoning events	Applicable calls whose reasoning text was labelled	712
Form rewrite	Contexts with at least two earlier range reads through the shell (shell $\rightarrow$ API) or two read_file calls (API $\rightarrow$ shell), all of which are rewritten	60 and 60 contexts in Table 1, 30 per direction in each earlier run
Result rewrite	Contexts with an earlier search whose output can be rewritten. The last search output in the context is rewritten	90 and 19 contexts in Table 1, 30 and 19 in the earlier run
State push	The first contexts of the result rewrite from hits to no matches, in their original version	40 contexts, three continuations per context and condition

B REPLAY AND THE OPERATION CLASSIFIER

Every state of a recorded call comes from a replay of the tokens the model produced, and every operation label from fixed rules.

B.1 TOKEN RECOVERY AND REPLAY

A teacher-forced replay needs the exact tokens the model produced, so the first step is to recover them. HARMONYAGENT records token ids. AGENTMOE reports its tokens in decoded surface form, which we re-encode. CODEX speaks the Responses API and is reconstructed from the event stream with an error of a few tokens per call.

The seed family was recorded through the Responses API and re-rendered locally with the server's own renderer, openai_harmony, whose layout we read back from ground-truth prompts. The system message carries the identity and the date, the developer message holds only the function namespace,

the instructions follow as a separate system message, and the conversation comes last. States were extracted only after the local rendering matched the server's exactly on a validation sample of every seed-family harness.

The replay stores one layer-20 state every eight generated tokens, starting at the first generated-token position. The layer-20 state is the output of block 20, counting blocks from zero, which the transformers library returns as hidden_states[21] because it counts the embedding first. Every decoder in the main text reads the single state at the first generated-token position, and so does every axis comparison at a call. Only the whole-run axis comparisons of Section 4, the Axes of rows 1–4 of Table 1, average over the tokens a harness has the model write (Appendix B.3). The first generated-token position is the one defined in Section 3. When the history control re-extracted these states from the raw replays, the frame coordinates it computed matched the stored ones exactly. The push runs read the state at the same position and block while they generate, and the rewrite runs were replayed to read it there.

B.2 THE OPERATION CLASSIFIER

Fixed rules map each classifiable tool-call attempt to one of five operations, and each operation can be written in API-style or in shell form (Table 4). The rules are applied in order: heredocs, then searches, sed and perl, the cat family, redirections and file moves, listings, and finally execution. A sed or perl command with an in-place flag is an edit, and any other sed or perl command is a read. A cat-family command that contains << or apply_patch is an edit, and shell reads count as reads only without an output redirect. Calls that fit no rule are excluded. An unregistered API-style name is mapped into the operation its name implies and keeps its API-style interface, so the API column of Table 4 counts attempts, registered or not. Registration validity is recorded separately. The adoption counts of Section 7 include registered tools only, the rewrite analyses report unregistered attempts separately, and in the anchor comparisons an unregistered next call enters neither half of the classifiable comparison.

Table 4: **Operations and the two interfaces that carry them.** Counts are TOOLS calls of the merged set under the exec treatment of the heredoc family. The API column counts attempts, including unregistered names, which Table 19 leaves out (Appendix A.4).

Operation	API tool	Shell	Shell	API
search	grep_search, search_file(s), search	grep, rg, ag, ack, find … -name	291	229
read	read_file, open_file	sed or perl without an in-place flag, cat, head, tail, less, more, nl, wc, od, xxd, without a redirect	226	273
edit	edit_file, write_file	sed -i, perl -i, >, >>, tee, patch, cat <<, mv, cp, rm, touch, mkdir	35	78
exec	run_file	python, pytest, make, bash, git, pip, tox, nox, ./	168	1
list	glob_files, find_file	ls, tree, du, df	50	13

Commands arrive in different shapes across harnesses, so the extractor scans for the command instead of matching a single pattern. TOOLS passes a command string, CODEX a cmd string that wraps bash -lc "…", HARMONYAGENT an argv array, and AGENTMOE a fenced block. The extractor scans rather than pattern-matches, because commands contain both brackets and escaped quotes, and it strips the bash -lc wrapper leniently, because the model often emits unclosed JSON.

The heredoc family, commands of the form python - <<'PY' … p.read_text() … PY, is ambiguous between two operations: a heredoc that prints information is a read by purpose and an execution by mechanism. Its treatment is a switch, and we report the main anchor result under every setting. The exec treatment keeps it an execution. It is used throughout, for the stored labels, every decoder, the anchors and Table 4, except in the adoption counts of Section 7 and Table 19, which use the read treatment and make it a read, and in the readout comparison of Table 7, which also uses the read treatment. The isolate treatment removes the family. Over all next calls at the anchor, the difference in API share between TOOLS and SEED is +40.8 points under the exec treatment, +40.6 under the read treatment and +40.8 under the isolate treatment, and all 24 non-tied tasks agree in sign each time ($p = 1.2 \times 10^{-7}$). No operation passes Holm under any of them. The isolate treatment

moves only the *exec* row, from −0.2 points (8 of 22, $p = 0.29$) to −2.4 points (4 of 18, $p = 0.031$). The read treatment moves the *exec* row in the same way and also the *read* row, from −3.7 to −1.0 points, because the heredocs it relabels become reads. Rows computed under different treatments are therefore never combined.

B.3 Which generated tokens count as writing

The axis comparisons of Section 4 average the state over the tokens a harness has the model write, and a channel rule decides which tokens count. Harmony places generated tokens in an analysis channel, in the arguments of tool calls and in a final reply, while AgentMoE writes plain text. The written rule, used for every reported axis comparison, keeps everything generated except tool arguments and the final reply. This rule is the same for every protocol but is not symmetric, because the command of AgentMoE sits inside its fenced block and is counted, while the command of a Harmony harness sits in the tool arguments and is not. The prose rule therefore also removes command spans from every harness by one textual rule: anything inside a fenced block, and anything from a to= recipient marker to the next <|call|>. Table 9 reports both rules where they differ.

C Readouts

We read the state through one fixed, released frame and check what it reads against a second readout and against simpler ones.

C.1 The frame and its named axes

The J64 frame is a released linear basis of 64 axes for the layer-20 residual stream of gpt-oss-20b, and we never refit it. A state h has coordinates $c = F^{+}(h - \mu)$, where F and μ come with the release. The release marks 16 axes as citable by name, namely axes 5, 6, 10, 21, 22, 26, 27, 33, 34, 35, 45, 46, 50, 52, 53 and 63, and all Holm corrections for named-axis comparisons use this fixed family of 16 axes. Other multiple-testing families are specified with their respective analyses. Axis values are reported in units of a pooled task-level SD, so that differences on different axes share one scale.

To read an axis in agent terms we list the words the model is about to write when the axis is high or low (Table 5). The listing pools the three hand-written harnesses, 138 248 states in all, and it is descriptive: a word that co-occurs with a projection does not make the axis mean that word. Each state is grouped by the token the model writes next, which the state has not yet seen. Axis 9 carries a label in the release but is not one of the 16 named axes. It appears because it leads the length control of Appendix D.1.

Table 5: **Words the model is about to write at the two ends of the axes that carry results.** Pooled over the three hand-written harnesses.

Axis	Name in release	High end	Low end	Meaning here
9	approach switch	approach, Instead, Use, Maybe, attempt	<|message|>, fences, _MODULE	switching approach
22	hold / hover	still, works, doesn't, keep, returns, fails	add, insert, create, modify, run	"it still does this" against acting
34	simplification drive	avoid, write, can, think, replace, try	contains, refers, trailing, indentation	intending against describing
53	caution / checking	ensure, confirm, consider, check, careful	contains, .path, _iter, includes	checking
6	uniqueness discussion	distinct, duplicates, ensure, Edge, can't	identifiers	edge-case checking
33	triangle geometry	triple, tuple, Patch, block, line, File	cmd, grep, sed, .exec, functions	patch structure against shell command

C.2 The agent lens

The second readout, the agent lens, is a Jacobian lens (Gurnee et al., 2026) that we built from replays of the seed family, so unlike the frame it is not independent of the harnesses we study. It maps the

layer-20 state through the Jacobian of the layer-23 state with respect to the layer-20 state, evaluated at 16 reference states and centred on the mean state over 4 000 agent positions, and it reads the result through 14 concept families. The reference states come from the three seed-family harnesses whose calls Section 5 decodes. We use the agent lens as a robustness check for operation decodability and for the asymmetry between the operation and the interface.

Later positions carry more of the operation, as expected once the model has begun to write (Table 6). Every decoder in the main text reads the first generated-token position, the earliest and hardest position in the table.

Table 6: **Operation decoding at later positions of the call** (agent lens, accuracy in % and normalised gain in points). The first generated-token position is the readout of the main text, the second uses every stored state before the call is revealed, and the third the last stored state before the reveal.

Harness	First position	All before the reveal	Last before the reveal
CODEX	60 / 35	65 / 43	67 / 46
SEED	56 / 26	65 / 41	65 / 42
TOOLS	54 / 25	61 / 37	65 / 43
HARMONYAGENT	54 / 29	59 / 36	63 / 42
EVOLVED	52 / 14	62 / 33	66 / 39
AGENTMOE	49 / 22	58 / 36	54 / 29

C.3 THE FRAME AGAINST SIMPLER READOUTS

The frame is a fixed measurement basis, not the source of what we decode (Table 7). All readouts are fitted and scored identically on the same 4 604 seed-family calls over 36 tasks, at the first generated-token position, with the same task-grouped folds and the same regulariser. They are the baseline set of Table 3, labelled in a separate pass under the read treatment, and the operation here takes six values. The random projection is redrawn in every fold, so a lucky basis cannot carry the result, and it matches the frame. What the frame adds is a basis that never changes, axes with names and a shared scale, which the axis comparisons of Section 4 and Section 6 need.

Table 7: **The frame against simpler readouts.** Normalised gain in points over a majority class of 44.8%, on the baseline set (Appendix A.4). For the random projection the gain is the mean ± SD over 10 draws, and the accuracy is the one this mean gain implies.

Readout	Dimension	Accuracy (%)	Gain
J64 frame	64	58.8	25.4
Random projection	64	58.8	25.2 ± 1.5
Raw residual stream	2880	56.9	21.8
Bag of the first generated token ids	400	54.4	17.3

The state readout exceeds the token-bag baseline by 8.1 points of normalised gain, with accuracies of 58.8% against 54.4%. The bag ignores order, keeps only the 400 most common token ids and sees none of the context. Its tokens are written after the state is read, which favours the bag.

Call by call, the state is right where the token readout is wrong more often than the reverse. On the 4 542 calls of the case-study set, the state alone is right 840 times and the token readout alone 637 times, an imbalance of 1.32 to one (McNemar exact $p = 1.4 \times 10^{-7}$). On this set the call is revealed a median of 53 generated tokens after the readout. Calls near that median show what the state carries. In scikit-learn-26194 the state is read at the first token of <|channel|>analysis<|message|>Now, we need to. The state reads *exec*, the token readout reads *search*, and 58 tokens later the model runs pytest -q. In xarray-3095 the first generated tokens name a line number, which points a token readout at *read*, while the state reads *search*, and 79 tokens later the model issues a grep.

D HARNESS COMPARISONS AND THE COMPLETE EVIDENCE TABLE

Table 8 is the complete version of Table 1. It adds the test, the p value and the null or control behind every cell, and the sections that follow take its rows in order.

D.1 THE STATE ACROSS THE HAND-WRITTEN HARNESSES

Harnesses that differ in protocol separate the state on more named axes than harnesses that share one. Between CODEX and HARMONYAGENT, which share Harmony, 5 of the 16 named axes pass Holm (Table 9). Between AGENTMOE and CODEX 7 pass, and between AGENTMOE and HARMONYAGENT 10. The median absolute difference per named axis is 0.361 SD within the protocol and 0.677 and 0.769 SD across it, while the pairs of the seed family differ by 0.08–0.14 SD. These are whole-trajectory averages in the pooled SD of each comparison, and Appendix F.4 explains why they cannot be set against the medians at an anchor.

The length and split-half checks show that the difference between CODEX and HARMONYAGENT does not come from how much each harness writes or from noise. CODEX writes 2.08 times as many tokens per task as HARMONYAGENT, yet an intercept at equal length is larger than the raw difference on 8 of the ten leading axes. On axis 9 it moves from −1.07 to −1.53 SD. Interleaved halves of the calls within each task give per-axis differences that correlate at 0.655–0.888 across the leading axes.

D.2 OPERATION MIX AND API SHARE

Within one protocol, harnesses differ far more in which interface carries an operation than in which operation they perform (Table 10, Table 11). Between harnesses that share a protocol the operation shift is 5.8–8.7 points, while the API share differs by 43.5–52.6 points wherever the menus differ. AGENTMOE, the only plain-text harness, edits more and searches less than the others, so its operation shift against HARMONYAGENT and CODEX is 15.9–21.1 points. Against CODEX its API share differs by only 5.9 points, because CODEX sends almost every operation other than an edit through the shell.

D.3 RESPONSE TO A TOOL ERROR

A trajectory mean cannot separate a harness that changes how the model processes its context from one that adds the same offset to every state, so we also compare how the state of each harness responds to a tool error. Inside each harness H and for each axis j we take the mean coordinate after a tool output that reported an error minus the mean after one that did not, and compare harnesses by the difference of these responses, in which any constant offset cancels:

$$R_H(j) = \overline{c_j}\,[\text{previous tool output reported an error}] - \overline{c_j}\,[\text{it did not}], \qquad \Delta R(j) = R_A(j) - R_B(j). \tag{1}$$

The error label comes from one fixed list of error signatures applied to the tool output the model read. Calls with fewer than 25 generated tokens, which are mostly replies that abandon a task, are excluded from every harness.

The hand-written harnesses respond to an error differently, while the seed family responds as one (Table 12). On axis 33 the state of HARMONYAGENT rises after an error while that of AGENTMOE falls, and CODEX differs from HARMONYAGENT on the same axis. Both contrasts pass Holm and exceed a null built from random halves within each harness. Between CODEX and AGENTMOE one axis, axis 50, exceeds the null threshold and fails Holm. Within the seed family the Holm-adjusted p is 1.000 on every named axis for every pair.

E WHAT THE STATE ADDS TO THE CALL HISTORY

The state adds to the call history for the operation and almost nothing for the interface, and the operation it carries transfers across harnesses and across interfaces.

Table 8: **Evidence ledger: the interface moves with the harness and the form of earlier reads, the operation with what the last search returned.** One row per tested comparison, favourable or not, in points unless marked. Sign tests are paired and two-sided, with ties excluded. Holm is step-down, with the family in brackets. Where no test is named, p comes from a permutation, $p = (k{+}1)/(B{+}1)$. Slashes follow the first column.

Comparison	Unit, n	Effect	Test, p	Null or control
What does a harness change? (Section 4; Table 11)				
Hand-written pairs, named axes	task, 50	5 of 16 within a protocol; 7 and 10 across	sign, Holm [16]	seed family: at most 1 of 16
TOOLS vs SEED, whole runs	task, 36	API share +43.5; operation TV 6.5	descriptive	EVOLVED vs SEED: TV 5.8
Resolve rate, TOOLS vs SEED	instance, 29	10 vs 4 resolved (8:2)	exact McNemar, 0.11	EVOLVED vs SEED: 0.73
What does the state add to the call history? (Section 5)				
Operation from the state, TOOLS	call, 1 364	gain 34.4; 30.6 on the 1 334 with a previous call	CV by task	frame 25.4, random 25.2 on 4 604 calls
Increment over the call history, TOOLS: operation / interface	call, 1 334	+11.8 [8.2, 15.5] / +1.0 [−0.1, 2.1]	state shuffled within task, 0.001 / 0.009	HARMONYAGENT +4.0 / 0.0; lens +8.1 / −0.4
TOOLS tool name / CODEX interface	call, 1 286 / 1 973	+2.1 [+0.4, +3.9] / +0.3	same, 0.001 / 0.001	tool name given the operation: −0.8
Transfer across harnesses	call, 910 to 2 023	gain 9 to 34 in all 20 cells among Harmony harnesses	permutation, all 0.002	AGENTMOE: 3 of 10 cells
Transfer across interfaces, TOOLS	call, 80 per class	32.2 across vs 37.3 within	permutation, all ≤ 0.013	four operations: ratio 0.50
After a shared file read, what differs? (Section 6; Table 1); TOOLS vs SEED				
Anchor, call $t{+}1$	task, 32	API share +45.8 (24 of 24); TV 4.4; each operation $\lvert\Delta\rvert \leq 4.4$	sign 1.2×10^{-7}; TV 0.61; each operation, Holm ≥ 0.84 [5]	EVOLVED vs SEED: interface 0.0, TV 4.3
Anchor, state at call t / $t{+}1$	task, 32	axes 3 / 1 of 16; decoder TV 3.5 / 4.9	Holm [16]; decoder 0.31 / 0.079	$\Delta P(\textit{search})$ at $t{+}1$: −4.7
Strict anchor, call $t{+}1$	task, 15	API share +11.6 (7 of 7); TV 10.1; decoder 4.0; axes 0 / 0 at t / $t{+}1$	sign 0.016; decoder 0.49	axis CI half-width 0.82 SD at $t{+}1$
Search anchor, call $t{+}1$	task, 13	API share +53.2 (9 of 9); TV 12.2; decoder 12.1	API (perm.) 0.0034; TV 0.52; decoder 0.061	EVOLVED: decoder 15.1, 0.018

Table 8 (continued).

Comparison	Unit, n	Effect	Test, p	Null or control
When is an offered tool used? (Section 7)				
API use, all vs applicable, TOOLS	call	search 38.9 vs 73.7%; read 44.9 vs 56.4%	descriptive	220 of 291 searches compound
Call history, applicable calls	call, 369 / 343	API use 89.7 vs 33.2% after an API vs a shell call	Mantel–Haenszel by task, 0.15	OR 17.5 pooled, 1.45 by task
Unregistered names, TOOLS vs SEED	task, 25 / 27	12 vs 1 tasks; 2.01 vs 0.04% of calls	Fisher exact, 2.6×10^{-4}	EVOLVED 3 / 29, CODEX 1 / 47
Name correctness from the state	call, 594	AUC 0.592	CV; label permutation	null 0.504 ± 0.060
Stated reasons for the interface	event, 712	stated in 0%	model labels, forced choice	—
Can the two halves be moved separately? (Section 8; Table 1)				
Form rewrite, shell $\rightarrow$ API / API $\rightarrow$ shell	context, 60 / 60	TV 8.6 / 2.5; API share +40.8 / −36.2; decoder 2.2 / 0.8	TV 0.046 / 0.79; API 1.0×10^{-4} / 1.0×10^{-4}; decoder 0.031 / 0.51	TV null$_{95}$ 8.4 / 7.8
Result rewrite, hits $\rightarrow$ no matches	context, 90	TV 29.0; API share −6.5; decoder 18.1	TV 1.0×10^{-4}; API 0.035; decoder 1.0×10^{-4}	TV null$_{95}$ 12.0
Result rewrite, no matches $\rightarrow$ hits	context, 19	TV 12.1 on 70 / 63 calls; API share −7.8; decoder 7.6	TV 0.10; API 0.50; decoder 0.029	TV null$_{95}$ 13.3
First form-rewrite run (aggregate counts)	generation, 480	TV 4.7 / 5.7; API share +40.6 / −24.3	operation χ^2 0.59 / 0.41; API (Fisher) 1.4×10^{-8} / 5.7×10^{-4}	—
Push toward *read*, α = −2 / −4	context, 40	TV 7.6 / 6.9; API share −3.7 / +0.8; decoder 47.4 / 55.3	TV 0.30 / 0.36; API 0.36 / 0.87; decoder 1.0×10^{-4} / 1.0×10^{-4}	TV null$_{95}$ 10.9 / 12.1; random push: Fisher 0.77 / 0.77
Push toward *search*, α = +2 / +4	context, 40	TV 9.2 / 12.1; API share −0.4 / +49.6 on 84 / 21 calls; decoder 48.2 / 60.0	TV 0.12 / 0.50; API 0.89 / 1.0×10^{-4}; decoder 1.0×10^{-4} / 1.0×10^{-4}	TV null$_{95}$ 10.7 / 23.7; random push: Fisher 0.53 / 0.60
Push, every dose, $\lvert\alpha\rvert$ = 1, 2, 4 and 8	context, 40	TV 3.9–12.1, within its null at every dose; decoder 29.1–63.5; 0 classifiable calls at α = +8	TV $p \geq 0.12$; against a random push, Fisher $p \geq 0.53$, Holm 1.00 [7]	random push of the same norm: decoder TV ≤ 1.5, TV beyond its null at 2 of 8
Push along frame axis 33	one push	broke the output format before changing behaviour	validity check	no interface target

Table 9: **Named axes that separate CODEX from HARMONYAGENT.** Δ = CODEX − HARMONYAGENT in pooled task-level SD under the written rule (Appendix B.3), the number of the 50 tasks with $\Delta > 0$, and the Holm-adjusted p over the 16 named axes. Between AGENTMOE and CODEX 7 axes pass under the written rule and 8 under the prose rule, and between AGENTMOE and HARMONYAGENT 10 pass.

Axis	Name in release	Δ/SD	Tasks with $\Delta > 0$	Holm
26	intersection operation	−0.90	11	0.0014
21	sameness comparison	−0.49	13	0.014
22	hold / hover	+0.83	37	0.014
34	simplification drive	−0.83	13	0.014
52	summation	+0.27	36	0.031

Table 10: **What each harness does and through which interface.** Left, the share of classified calls by operation (%). Right, the API share of each operation (%), on the merged set.

	Operation mix					
Harness	*search*	*read*	*edit*	*exec*	*list*	API share by operation
SEED	34	40	7	15	4	none, no API tool is registered
TOOLS	38	37	8	12	5	*search* 44, *read* 55, *edit* 69, *list* 21, *exec* 1
CODEX	39	34	6	14	6	*edit* 88, the rest almost only through the shell
HARMONYAGENT	36	30	14	16	4	*search* 61, *read* 73, *edit* 79, *list* 78, *exec* 0
EVOLVED	29	43	8	14	5	almost none, unregistered attempts only
AGENTMOE	21	34	22	19	4	none, plain-text protocol

E.1 THE HISTORY CONTROL

The history control asks how much better the operation and the interface of a call are predicted when the state is added to a model of the calls that came before it. The history model sees the previous call's operation, with an extra class for calls that fit no operation, the previous call's interface, the interaction of the two, the API share over the last three earlier tool calls and over all earlier ones, and the position of the call in its trajectory. The history is rebuilt from the full trace of the task, so it includes earlier calls that never received a state or a class. Every call that follows at least one earlier tool call enters, which gives 1 334 calls in TOOLS and 903 in HARMONYAGENT.

The state enters through its own decoder, so a small increment cannot be blamed on the extra parameters. Within each training fold a state-only decoder is cross-fitted to produce out-of-fold log-probabilities, and a second logistic regression combines these log-probabilities with the history features. Both stages are standardised on the training fold and fitted with $C = 0.5$ under five task-grouped folds. The null permutes the state rows within each task 1 000 times while keeping the history, so information about the task survives in the null and information about the single call does not. Intervals come from 2 000 task-level bootstrap resamples of the out-of-fold predictions. A model that adds the raw coordinates to the history directly overfits and gives smaller increments for both halves.

The state adds to the call history for the operation under every readout and in both harnesses, and it adds almost nothing for the interface (Table 13). In TOOLS under the frame, the accuracy for the operation rises from 48.4% with the history alone to 60.3% with the state, an increment of +11.8 points. For the interface it rises from 79.4% to 80.4%, an increment of +1.0 points, and the AUC of the interface moves from 0.870 to 0.879. The agent lens and the random projection show the same asymmetry, and so does HARMONYAGENT.

The one exception is small. Under the frame the state lowers the log-loss of the interface in TOOLS by 0.010 nats per call, beyond its null ($p = 0.001$), against 0.130 for the operation. Under the agent lens and the random projection the interface increment stays inside its null ($p = 0.24$ and $p = 0.64$), and in HARMONYAGENT it stays inside its null under every readout. With stronger regularisation, $C = 0.05$, the frame's increments in TOOLS are +12.8 points for the operation and +1.2 points for

Table 11: **Within a protocol the operation moves little and the interface moves with the menu; across protocols the operation moves as well.** Whole runs of each pair over their shared tasks, without one-to-one matching of calls, measured as in rows 1–4 of Table 1. These comparisons are descriptive. Each pair takes two lines: the first harness with its API share, then the second; estimates sit above their 95% bootstrap intervals over tasks. *Interface*: Δ API share, first minus second. *Operation*: TV between the two operation distributions. *Decoder*: TV between the mean posteriors of the frozen decoder, which was fitted on all six harnesses; *refit*: the same recipe fitted on the other four harnesses only. *Null*: 95th percentile of a null that re-splits each task's pooled calls; it ignores the dependence among calls of one run, so it is a lower reference, and no entry is marked. *Axes*: named axes passing Holm, out of 16; *median*: median $|\Delta|$ over those axes, in pooled task-level SD. Points throughout, except API share (%) and the median (SD).

		Interface	Operation		Decoder			Axes	
Pair	API share	Δ API share	TV	Null	TV	Refit	Null	k	Median
Seed family, 36 shared tasks									
TOOLS	43.5	+43.5	6.5	5.0	2.8	3.4	2.4	0	0.10
vs SEED	0.0	[+32.8, +54.4]	[3.2, 11.4]		[1.5, 5.9]				
TOOLS	43.5	+43.5	8.7	4.8	5.6	5.3	2.7	1	0.14
vs EVOLVED	0.1	[+33.0, +54.1]	[5.7, 14.2]		[3.1, 9.0]				
EVOLVED	0.1	+0.1	5.8	4.3	3.1	3.6	2.3	0	0.08
vs SEED	0.0	[0.0, +0.2]	[3.1, 11.5]		[1.4, 6.4]				
Hand-written harnesses, 50 shared tasks									
CODEX	5.9	−52.6	8.7	5.9	9.0	9.4	3.8	5	0.36
vs HARMONYAGENT	58.5	[−59.6, −45.2]	[5.0, 14.4]		[6.3, 12.3]				
AGENTMOE	0.0	−58.5	15.9	7.2	15.5	23.8	3.9	10	0.77
vs HARMONYAGENT	58.5	[−65.2, −51.3]	[10.0, 22.9]		[11.8, 19.5]				
AGENTMOE	0.0	−5.9	21.1	6.5	21.7	20.0	3.7	7	0.68
vs CODEX	5.9	[−7.2, −4.5]	[16.0, 29.0]		[17.3, 26.5]				

Table 12: **Response to a tool error**, Eq. (1). R is in pooled SD and n is the number of paired tasks. The null is the 95th percentile of the same statistic over random halves within each harness.

Contrast	Axis	n	R_A	R_B	ΔR/SD	Holm	Above null
HARMONYAGENT – AGENTMOE	33 triangle	35	+1.06	−0.39	+1.44	0.0019	yes
HARMONYAGENT – AGENTMOE	53 caution	35	−0.31	+0.74	−1.05	0.0076	yes
HARMONYAGENT – AGENTMOE	22 hold	35	−0.84	+0.19	−1.03	0.026	yes
CODEX – HARMONYAGENT	33 triangle	22	−0.57	+1.37	−1.94	0.014	yes
CODEX – AGENTMOE	none	24					axis 50

the interface (p = 0.001). With the heredoc family under the read treatment instead of the exec treatment of the stored labels (Appendix B.2), they are +10.9 and +0.7 points.

The operation increment is positive in all six harnesses. Under the frame it is +14.6 points in CODEX, +13.2 in SEED, +10.7 in EVOLVED and +4.5 in AGENTMOE, each with p = 0.001. SEED, EVOLVED and AGENTMOE register no API tool, and CODEX sends almost every call through the shell, so its history alone already predicts the interface with 94.0% accuracy and the state adds +0.3 points.

Once the history and the current operation are known, the state does not add to the prediction of which API tool is named. On the 1 286 TOOLS calls with a tool-name label, the increment of the frame is −0.8 points (p = 0.65).

E.2 THE INTERFACE WITHIN ONE OPERATION

Inside one operation the state separates API calls from shell calls only as far as the previous call already does. Among the 520 searches of TOOLS, the frame's interface decoder reaches an AUC of 0.75. Within strata of the previous interface the same AUC falls to 0.645. The history alone reaches

Table 13: **What the state adds to the call history.** Out-of-fold accuracy (%) of the history-only decoder and of the stacked state and history decoder, the increment in points with its 95% task-bootstrap interval, and the permutation p of the increment (state rows permuted within task, 1 000 permutations). Increments are computed before rounding, so an increment can differ in its last digit from the difference of the rounded accuracies. The random projection is shown for the first of its ten bases throughout, and its mean increment over all bases is +12.0 for the operation and +0.3 for the interface in TOOLS, and +3.3 and −0.4 in HARMONYAGENT. $C = 0.5$. The interval resamples tasks, while the permutation p shuffles states within tasks, so an interval that includes zero can accompany a small p.

		Operation				Interface			
		Hist.	+State	Increment	p	Hist.	+State	Increment	p
TOOLS	frame	48.4	60.3	+11.8 [8.2, 15.5]	0.001	79.4	80.4	+1.0 [−0.1, 2.1]	0.009
	lens	48.4	56.5	+8.1 [5.3, 11.0]	0.001	79.4	79.0	−0.4 [−0.9, 0.1]	0.99
	random	48.4	60.2	+11.8 [8.2, 15.6]	0.001	79.4	79.2	−0.1 [−0.5, 0.2]	0.94
HARMONYAGENT	frame	54.2	58.1	+4.0 [1.3, 6.6]	0.001	73.5	73.5	0.0 [−1.0, 1.0]	0.50
	lens	54.2	56.5	+2.3 [−0.2, 4.8]	0.001	73.5	72.9	−0.7 [−1.6, 0.2]	0.96
	random	54.2	57.3	+3.1 [0.6, 5.6]	0.001	73.5	73.3	−0.2 [−1.5, 1.0]	0.68

0.91, and adding the state to it changes the fold-mean AUC by +0.002 ($p = 0.39$). Among the 499 reads the state alone reaches 0.72, within strata 0.640, the history alone 0.92, and the state changes the history's AUC by −0.005 ($p = 0.86$). The previous interface matters on its own: a search goes through grep_search 74% of the time after an API call and 14% after a shell call, and a read goes through read_file 88% and 31% of the time.

E.3 HOW FAR AHEAD THE READOUT IS

The readout is taken well before the call is written. In TOOLS the call is revealed a median of 45 generated tokens after the first generated-token position, with quartiles 21 and 164, and across the six harnesses the medians span 29–63 tokens, shortest in AGENTMOE and longest in HARMONYAGENT, with 48 pooled over all 8 172 calls. The reveal is the first token of the call, the recipient marker to= in Harmony and the opening fence for AGENTMOE. The distribution is skewed to the right, with a mean of 239 tokens in TOOLS. API calls are revealed sooner than shell calls (medians 35 and 52), reads soonest (24) and edits latest (251).

E.4 TRANSFER BETWEEN HARNESSES

A decoder fitted on one harness reads the operation on the others, and the zero point of this test is a permutation null rather than zero (Table 14). The decoder is fitted on all calls of the source harness, with coordinates standardised by the pooled mean and SD over the six harnesses, and applied unchanged to the target. Each harness contributes 737 to 2 023 classified calls. Gains are normalised by the target's majority class. A decoder fitted on shuffled labels scores below that majority class, so the null means of the cells range from −32.6 to −1.6. Each cell is tested with 500 label shuffles and the add-one estimator, whose floor is 0.002.

Every cell among the Harmony harnesses exceeds its null (20 of 20), and 3 of the cells that involve AGENTMOE do. Those are AGENTMOE to EVOLVED ($p = 0.002$), AGENTMOE to SEED ($p = 0.016$) and HARMONYAGENT to AGENTMOE ($p = 0.018$). The cell from AGENTMOE to EVOLVED gains 2.2 points against a null of −32.6 ± 19.3, a success that a test against zero would hide. The cell from TOOLS to AGENTMOE scores −22.1 points against a null of −18.0 ± 7.0 ($p = 0.74$), so it is indistinguishable from a decoder fitted on noise.

Table 14: **Transfer of the operation decoder** (frame, first generated-token position, normalised gain in points). Rows are fitted and columns tested; the diagonal is in-sample. The cross-validated gain of each target gives the own point of Figure 3b as its mean over the Harmony harnesses. In the lower block every operation class is subsampled to 26 calls on both sides (mean of 20 resamples, Figure 6); its diagonal would be in-sample on the subsample and is left blank.

Fitted on \ tested on	SEED	EVOLVED	TOOLS	HARMONYAGENT	CODEX	AGENTMOE
As fitted						
SEED	42	24	34	21	26	−18
EVOLVED	32	37	27	9	21	−8
TOOLS	34	24	47	21	24	−22
HARMONYAGENT	21	11	23	45	18	−4
CODEX	26	16	29	13	49	−31
AGENTMOE	2	2	−1	−8	−7	46
Cross-validated on target	29.3	20.3	34.4	31.2	40.4	21.1
Classes balanced						
SEED	–	35	32	22	30	3
EVOLVED	42	–	33	17	32	6
TOOLS	39	34	–	21	28	4
HARMONYAGENT	22	20	19	–	20	2
CODEX	36	28	27	17	–	4
AGENTMOE	12	11	12	6	8	–

Transfer recovers most of what a decoder fitted on the target achieves. Over the 20 cells among the Harmony harnesses, the median ratio of the transferred gain to the target's own cross-validated gain is 0.69, with a range of 0.29–1.18. The ratio of the mean transferred gain to the mean cross-validated gain, which Figure 3b shows, is 73%. Transfer does not depend on seeing the same tasks: the eight cells within a family, which share their tasks, average 25.7 points, and the twelve cells across families, which mostly do not, average 20.6.

Transfer does not ride on the wire format either. We took the same content verbatim and rendered it once in Harmony tokens and once in the plain THOUGHT: and fenced bash format of AGENTMOE, on 450 paired states. Against a majority class of 41.1%, the decoder reaches 60.0% from Harmony to Harmony (gain 32.1), 58.4% from plain to plain (29.4), 59.8% from Harmony to plain (31.8) and 54.9% from plain to Harmony (23.4). The same content in the other format moves the state by a median of 0.15 SD per axis, with a largest shift of 0.64. The separation between AGENTMOE and the Harmony harnesses measured the same way is 0.82–1.01 SD.

E.5 CLASS BALANCE AND AGENTMOE

Part of the weak transfer involving AGENTMOE comes from its different mix of operations. AGENTMOE edits in 22% of its calls where the others edit in 6–14%, and searches in 21% where the others search in 29–39%. Balancing the class counts of both sides over repeated resamples raises the mean of the cells that leave AGENTMOE from −2.3 to 9.6 points, the mean of the cells that enter it from −16.7 to 3.7, and the mean of the other cells from 22.6 to 27.6 (resampling SD 4.8, 3.1 and 2.5); balancing lowers 3 of the other cells. Balancing thus lifts every AGENTMOE cell above zero, but none reaches the lowest other pair (Figure 6). On the full balanced matrix (Table 14, lower block) no off-diagonal cell is negative, and the smallest is +1.9. The largest AGENTMOE cell, 11.9, stays below the smallest other one, 16.8.

A gap remains after balancing: the other cells still transfer three to seven times better than those involving AGENTMOE. The result does not depend on the readout. Under the agent lens the off-diagonal gains correlate with those of the frame at +0.885, the AGENTMOE group is negative under both (−9.4 and −9.5) and the rest is positive under both (19.1 and 22.6).

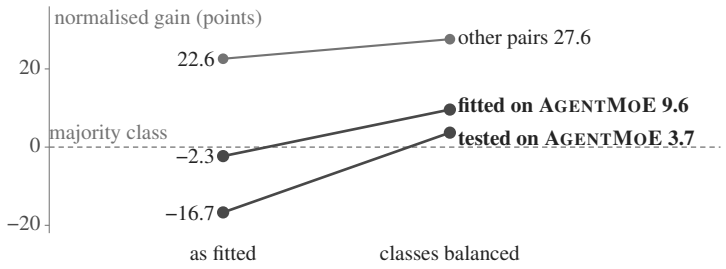


Figure 6: **Transfer to and from AGENTMOE before and after class balancing.** Group means of the normalised gain of the operation decoder fitted on one harness and tested on another, as fitted and with every operation class subsampled to 26 calls on both sides (mean of 20 resamples): the cells fitted on AGENTMOE and the cells tested on AGENTMOE (dark) and the 20 other pairs (grey); dashed: majority class.

E.6 TRANSFER ACROSS INTERFACES

Inside TOOLS, a decoder trained on calls made through one interface reads the operation of calls made through the other (Table 15). In TOOLS the same operations go through both interfaces in quantity, so the question can be asked within one harness. We restrict to *read* and *search*, sample 80 calls per class on each side, and hold out whole tasks so that no task appears on both sides. The mean gain within an interface is 37.3 points and across interfaces 32.2, a ratio of 0.87 (Figure 3b). The number of label permutations per cell ranges over 213–400 because the number of valid task splits differs between cells.

Table 15: **The operation decoder across interfaces inside TOOLS.** Normalised gain (points), the mean and SD of its permutation null, the add-one permutation p and the number of permutations. The middle rows train on one interface and test on the other.

Fitted on	Tested on	Gain	Null	p	Permutations
shell	shell	31.3	1.5 ± 11.6	0.005	400
shell	API	36.7	0.3 ± 12.8	0.0033	300
API	shell	27.8	−1.1 ± 11.3	0.013	313
API	API	43.2	0.1 ± 12.1	0.0047	213

With every operation that both interfaces share, the comparison loses power rather than changing direction. TOOLS issues only twelve glob_files calls, so the version with *list* and *edit* added is capped at twelve calls per class. Its cells read 38.2, 20.4, 16.4 and 35.0 points in the order of Table 15, the ratio falls to 0.50, and the cell from API to shell is no longer significant (p = 0.14). At this size the null has an SD of 12–19 points.

E.7 THE HARNESS SHIFT IN THE OPERATION SUBSPACE

The difference between harnesses shows no enrichment in the subspace that the operation decoder uses. For each hand-written pair we take the difference of the harness means in the frame, divide it by the shared SD, and measure the share of its squared norm that falls into the subspace spanned by the decoder's weights. The five weight vectors of a multinomial decoder sum to zero, so this subspace has dimension four. The share is 0.9–3.4% across the hand-written pairs, against 6.25% expected for a random subspace of the same dimension, with an SD of 4.3 points over 2 000 random draws, so we claim only the absence of enrichment and not a depletion.

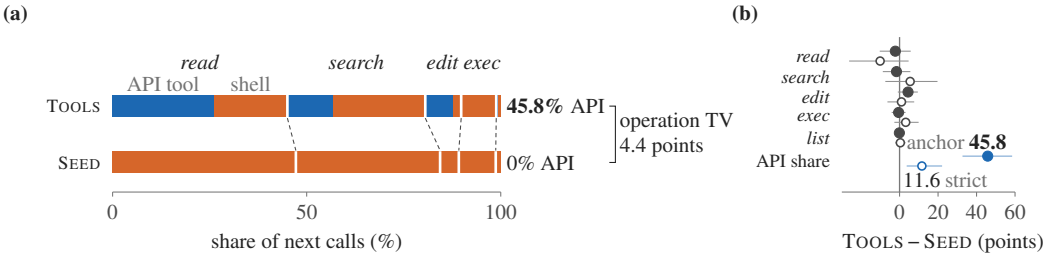


Figure 7: **The next call at file-read-matched anchors.** (a) Task-mean shares of the classifiable next calls of TOOLS and SEED at the anchor (32 tasks), by operation and interface; SEED registers only the shell. (b) TOOLS minus SEED per operation and in API share, at the anchor (filled) and at the strict anchor, where call t went through the shell in both harnesses (hollow, 15 tasks), with 95% task-bootstrap CIs.

F FILE-READ-MATCHED ANCHORS

At a file-read-matched anchor the operation distribution of the next call of TOOLS does not exceed its permutation null against a shell-only harness, while its interface differs widely, and this section builds the anchors and gives every comparison made on them.

F.1 HOW ANCHORS ARE BUILT

An anchor is a moment at which TOOLS and SEED have just read the same file of the same task, and the comparison is always made at the next call (Figure 7). For each task we take the files that both harnesses touch. Every call that reads such a file is an anchor call t, and call t+1 is recorded. A task enters when both harnesses have at least three such next calls. Shares are computed per task and averaged over tasks with equal weight, and the sign test is paired by task with ties excluded. The five operations form one Holm family, whose threshold for the smallest p is 0.01.

The anchors of Table 1 keep every next call in the task, and Table 16 gives their behaviour. The strict anchor starts from a slightly different selection, the anchor calls that carry a stored state and whose file is shared over the whole run, and adds the requirement that call t went through the shell in both harnesses. We also report this stored-state selection, because the strict anchor is its restriction. SEED registers no API tool, so the interface comparison measures how often TOOLS uses its tools where SEED can only use the shell. At the anchors the difference in API share between TOOLS and SEED is +40.8 points over all next calls and +45.8 points over the classifiable next calls, on which Table 1 computes both halves. Both counts use the exec treatment of the heredoc family. On the stored-state selection the difference is +47.6 points. In the negative control, EVOLVED against SEED, both harnesses register only the shell, so the strict restriction removes nothing for this pair.

F.2 THE NEXT CALL AT THE ANCHOR

At the next call no single operation exceeds its Holm threshold in either comparison with a shell-only harness, while the API share differs sharply in both (Table 16). After a shared file read, the operation distributions of TOOLS and SEED differ by 4.4 points in total variation, while TOOLS makes 45.8% of its next calls in API-style form (Figure 7a), and of the two halves only the interface exceeds its permutation null (Figure 7b). The single operations of TOOLS and SEED differ by at most 3.8 points over all next calls, the share that Table 16 counts, and by at most 4.4 points over the classifiable next calls, which Figure 7b draws and on which Table 1 computes the operation shift. That shift stays inside its permutation null (p = 0.61). At the strict anchor TOOLS still makes 11.6% of its next calls in API-style form; there *read* differs by −10.1 points and the operation shift of 10.1 points does not exceed its null (17.6), so that anchor cannot resolve operation differences of that size, and no single operation passes Holm at either anchor. The negative control shows the size of the operation differences that arise between two harnesses with the same menu, and its API shares are tied in every task. The other two treatments of the heredoc family move only the *exec* and *read* rows (Appendix B.2).

The interface difference at the anchor tracks how TOOLS made the shared read. Averaged over tasks, its next call uses an API tool 64.8% of the time after a read_file read (23 tasks) and 14.0% after a shell read (18 tasks). Within the 9 tasks that have both kinds of anchor the difference is +15.8 points, positive in 6 of 9 (sign test, $p = 0.51$).

Table 16: **The next call after a shared file read.** For each comparison, the task-mean difference in the share of all next calls (points), the number of non-tied tasks with a positive difference, and the exact sign-test p with ties excluded, under the exec treatment of the heredoc family, on 32 tasks. The Holm threshold for the smallest of the five operation p values is 0.01. The last row is the operation shift of Table 1, the total variation over the classifiable next calls, with its permutation p.

	TOOLS − SEED			TOOLS − EVOLVED			EVOLVED − SEED		
	Δ	k of n	p	Δ	k of n	p	Δ	k of n	p
read	−3.7	16 of 31	1.00	−2.1	17 of 32	0.86	−4.6	14 of 31	0.72
search	−1.6	13 of 30	0.58	−2.9	17 of 32	0.86	+2.5	18 of 31	0.47
edit	+3.8	13 of 19	0.17	+1.9	13 of 23	0.68	+0.9	9 of 17	1.00
exec	−0.2	8 of 22	0.29	+1.1	10 of 19	1.00	−1.1	12 of 24	1.00
list	−0.1	5 of 9	1.00	−0.5	2 of 6	0.69	+0.1	3 of 8	0.73
API share	+40.8	24 of 24	1.2×10^{-7}	+39.7	24 of 24	1.2×10^{-7}	0.0	all tied	1.00
Operation shift (TV)	4.4		0.61	3.4		0.84	4.3		0.59

F.3 THE SEARCH ANCHOR

When both harnesses have just searched the same file, the behaviour at the next call has the same shape as at the anchor. Over all next calls, the difference in API share is +47.3 points between TOOLS and SEED (9 of 9 non-tied tasks, $p = 0.0039$) and +40.4 points between TOOLS and EVOLVED (9 of 9, $p = 0.0039$), and the negative control is tied in every task. On the classifiable next calls, which Table 8 counts, the difference against SEED is +53.2 points. The largest operation difference is in *read*: −14.5 points against SEED (5 of 12, $p = 0.77$) and −19.8 against EVOLVED (3 of 14, $p = 0.057$). The negative control, EVOLVED against SEED, itself differs by +10.0 points in *read* ($p = 0.21$), so we count this difference neither as an effect nor as an absence. Anchors on *edit* and *exec* pair too few tasks to be tested. The decoder on this anchor is in Appendix F.5.

F.4 THE STATE AT THE ANCHOR CALL AND AT THE NEXT CALL

We read the 16 named axes twice: at call t, to check that the two harnesses enter the anchor alike, and at call t+1, where the comparison is made (Table 17). For each axis we take the task-mean difference between the harnesses in the pooled SD of the panorama and report the median absolute difference over the named axes. This median is compared with a null in which the two harnesses are swapped within each task, and the resolution of the design is the median half-width of the per-axis 95% intervals.

The anchors fail the check at call t, and the strict anchor passes it: once call t goes through the shell in both harnesses, no named axis passes Holm (Figure 8). At call t on the anchors, 3 named axes pass Holm against SEED and 2 against EVOLVED, and on the stored-state selection 3 and 3. In these anchors TOOLS often made the shared read through an API tool while SEED could only use the shell, and the strict anchor matches the interface of call t as well. On the strict anchor the medians of 0.159 SD against SEED and 0.234 SD against EVOLVED lie inside their nulls. On the axes that pass Holm at the anchor, counterpart moves from −1.56 to −0.59 SD and action from +1.36 to +0.08 SD, while addition does not shrink (−0.68 to −0.73 SD) and yet fails Holm on the strict anchor (adjusted $p = 0.23$). The other 13 named axes pass Holm for TOOLS minus SEED nowhere, at call t or t+1, and the control EVOLVED minus SEED passes none anywhere. The strict anchor has wider intervals (median per-axis 95% half-width 0.82–0.92 SD, against 0.55–0.59 SD at the anchor), so part of this null result reflects lower resolution.

At the next call the strict anchor again shows no axis passing Holm, while the anchors and the stored-state selection show a few. On the strict anchor at call t+1, 0 and 0 axes pass against SEED and EVOLVED, with medians inside their nulls ($p = 0.62$ and $p = 0.84$). On the anchors 1 and 3

axes pass, and the medians exceed their nulls (p = 0.025 and $p = 4.0 \times 10^{-4}$), and the stored-state selection behaves the same way. The negative control passes no axis anywhere. The strict anchor pairs only 15 tasks, so its per-axis resolution is 0.85–0.92 SD at call t and 0.82–0.85 SD at call t+1, against 0.54 SD at call t on the stored-state selection.

Table 17: **The named axes at the anchor call t and at the next call t+1.** Median absolute per-axis difference over the 16 named axes (panorama SD), the 95th percentile of the same median when the harnesses are swapped within tasks and its permutation p, the named axes passing Holm, the median half-width of the per-axis intervals (width), and the median over all 64 axes (all). The strict restriction leaves EVOLVED − SEED unchanged, so its rows are those of the stored-state selection.

Selection	Call	Comparison	Tasks	Median	Null	p	Holm	Width	All
anchor	t	TOOLS − SEED	32	0.346	0.321	0.03	3	0.55	0.407
		TOOLS − EVOLVED	32	0.281	0.329	0.14	2	0.58	0.436
		EVOLVED − SEED	32	0.193	0.241	0.22	0	0.48	0.237
	t+1	TOOLS − SEED	32	0.366	0.334	0.025	1	0.57	0.380
		TOOLS − EVOLVED	32	0.522	0.348	4.0×10^{-4}	3	0.59	0.370
		EVOLVED − SEED	32	0.190	0.240	0.21	0	0.45	0.234
stored state	t	TOOLS − SEED	32	0.431	0.315	0.0025	3	0.54	0.442
		TOOLS − EVOLVED	34	0.327	0.313	0.035	3	0.55	0.383
		EVOLVED − SEED	34	0.129	0.241	0.72	0	0.48	0.245
	t+1	TOOLS − SEED	32	0.373	0.328	0.017	2	0.55	0.386
		TOOLS − EVOLVED	34	0.420	0.344	0.0093	4	0.60	0.280
		EVOLVED − SEED	34	0.133	0.250	0.70	0	0.49	0.200
strict	t	TOOLS − SEED	15	0.159	0.429	0.95	0	0.85	0.186
		TOOLS − EVOLVED	15	0.234	0.458	0.80	0	0.92	0.252
	t+1	TOOLS − SEED	15	0.256	0.431	0.62	0	0.82	0.206
		TOOLS − EVOLVED	15	0.192	0.416	0.84	0	0.85	0.221

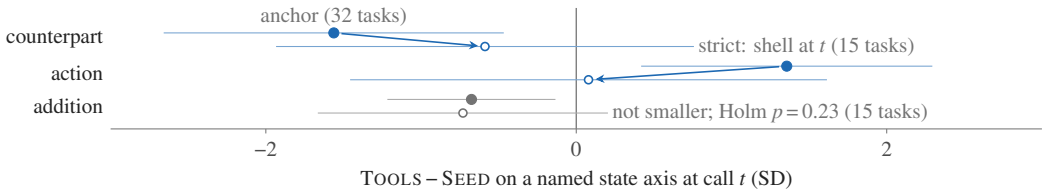


Figure 8: **Named axes at the anchor and at the strict anchor.** TOOLS minus SEED at call t, in panorama SD, on the named axes that pass Holm at the anchor (frame axes 5, 27 and 45, top to bottom), from the anchor (32 tasks) to the strict anchor (15 tasks); filled points pass Holm, hollow points do not; lines: simultaneous 95% t intervals over tasks, Bonferroni-corrected for the 16 named axes.

F.5 THE OPERATION DECODER AT THE ANCHORS

The operation decoder read at call t+1 exceeds its null in one comparison only, TOOLS against EVOLVED on the search anchor (Table 18). The decoder is fitted on 4 436 seed-family calls under task-grouped folds, so every anchor call is scored by a fold that has not seen its task. It holds out at 59.7% against a majority class of 40.3%, a gain of 32.6 points. We summarise it by the total variation between the task-mean posteriors of the two harnesses, which is on the same scale as the operation shift. On the anchors this distance is 4.9 points (p = 0.079), at the edge of its null, against a behavioural operation shift of 4.4 points (p = 0.61). On the strict anchor it is 4.0 points (p = 0.49), well inside its null. The frozen decoder of Section 8, fitted once on all six harnesses and therefore on these calls as well, gives 3.6 points (p = 0.21) on the anchors.

On the search anchor the state separates TOOLS from EVOLVED although their behaviour does not. The decoder's distance between them is 15.1 points against a null of 12.9 (p = 0.018), while their behavioural operation shift of 18.6 points stays inside its null (p = 0.10). Against SEED the distance

of 12.1 points lies at its null of 12.4 (p = 0.061), with a behavioural shift of 12.2 points (p = 0.52). The anchor pairs only 13 and 14 tasks, the p values are uncorrected, and no state was read at its call t.

Table 18: **The operation decoder and the behaviour at call t+1.** Total variation between the task-mean decoder posteriors of the two harnesses (points) with its 95% task-cluster bootstrap interval, its permutation p under within-task swaps of the harnesses and the 95th percentile of that null. Then come the behavioural operation shift on the same classifiable calls with its p and the difference in API share on those calls (points). Exec treatment of the heredoc family throughout.

			Decoder			Operation shift		
Selection	Comparison	Tasks	TV	p	Null	TV	p	ΔAPI
anchor	TOOLS – SEED	32	4.9 [2.0, 9.0]	0.079	5.3	4.4	0.61	+45.8
	TOOLS – EVOLVED	32	5.5 [2.5, 10.0]	0.064	5.7	3.4	0.84	+45.1
	EVOLVED – SEED	32	1.3 [0.8, 5.4]	0.85	4.1	4.3	0.59	0.0
stored state	TOOLS – SEED	32	5.1 [2.0, 9.3]	0.069	5.3	5.0	0.48	+47.6
	TOOLS – EVOLVED	34	3.9 [1.5, 8.6]	0.25	5.6	1.6	0.98	+43.1
strict	TOOLS – SEED	15	4.0 [1.9, 11.8]	0.49	7.9	10.1	0.42	+11.6
	TOOLS – EVOLVED	15	5.3 [2.4, 11.5]	0.21	6.8	16.9	0.11	+11.6
	EVOLVED – SEED	34	1.4 [0.7, 5.1]	0.80	3.9	3.7	0.76	0.0
search	TOOLS – SEED	13	12.1 [4.2, 24.1]	0.061	12.4	12.2	0.52	+53.2
	TOOLS – EVOLVED	14	15.1 [6.5, 25.0]	0.018	12.9	18.6	0.10	+44.1
	EVOLVED – SEED	16	5.5 [2.2, 12.1]	0.26	8.5	10.1	0.39	0.0

On the strict anchor the API share still differs, by +11.6 points against SEED and +11.6 against EVOLVED, and against SEED the difference has the same sign in 7 of 7 non-tied tasks (p = 0.016).

G WHEN AN OFFERED TOOL IS USED

Whether TOOLS uses an evolved tool depends on whether the tool could serve the call and on how the trajectory has been calling, and the tested decoder does not establish reliable prediction of whether the name the model reaches for is registered.

G.1 APPLICABLE CALLS

Counting adoption over the calls an evolved tool could have served changes the picture for *search* and barely changes it for *read* (Table 19). A shell call is applicable when a single API call would return the same content. For *search* this means a plain recursive grep without pipes, context lines or several files. Of the 291 shell searches of TOOLS, 220 are compound commands that grep_search cannot express, and 5 are find commands that glob_files serves and that are counted under *list*. For *read* it means sed -n 'A,Bp' over at most 400 lines, or a whole file. Of the 319 shell reads, 201 are applicable, 111 combine several commands, 5 span more than 400 lines and 2 are not reads of file content. These counts use the read treatment of the heredoc family (Appendix B.2).

Table 19: **Adoption of the evolved tools in TOOLS under two denominators.** API counts calls to registered tools only, which is why they are lower than the attempts of Table 4. Rates are the API share over all calls of the operation, API over API plus shell, and over the applicable calls, API over API plus applicable (%).

Operation	API	Shell	Applicable	Not applicable	Rate over all	Rate over applicable
search	185	291	66	225	38.9	73.7
read	260	319	201	118	44.9	56.4

G.2 THE CALL HISTORY

The interface of a TOOLS call follows the interface of the call before it, mostly because tasks differ and partly within tasks. On the 712 applicable calls, the next call goes through an API tool 89.7% of the time after an API call (n = 369) and 33.2% after a shell call (n = 343), a pooled odds ratio of 17.5. Stratified by task, the Mantel–Haenszel odds ratio falls to 1.45, with a 95% interval from 0.84 to 2.51 (p = 0.15). Only the 17 tasks in which both previous interfaces and both next interfaces occur inform this estimate. 27 tasks have applicable calls after both kinds of previous call, and these are the tasks that Figure 4b averages over. Within them the API share after an API call exceeds that after a shell call by 4.3 points on average, and it is higher in 8 of 17 non-tied tasks (p = 1.00). Some trajectories settle on the evolved tools early and stay there, while others keep the shell.

Over all 3 197 consecutive pairs of tool calls in 36 tasks, which include calls an evolved tool could not serve, a dependence within tasks does survive. The next call goes through an API tool 67.8% of the time after an API call and 22.9% after a shell call, a pooled odds ratio of 7.1. Stratified by task the odds ratio is 2.43 [2.05, 2.89] ($p = 1.3 \times 10^{-28}$), and stratified by task and by the tercile of the position in the trajectory it is 1.54 [1.29, 1.85]. Within a task the difference averages +25.3 points and is positive in 27 of 35 tasks (sign test p = 0.0019).

Over all pairs the dependence is strongest for reads. When the next call is a read, it goes through read_file 86.5% of the time after an API call and 38.1% after a shell call, with an odds ratio of 4.50 stratified by task and 2.84 stratified by task and position ($p = 4.6 \times 10^{-6}$). When the next call is a search the shares are 68.8% and 13.0%, and the odds ratios 3.24 and 1.71 (p = 0.052). As single predictors of the next interface over the 1 334 calls of the history control, the previous interface reaches an AUC of 0.769 and the API share over all earlier calls 0.85. On the 712 applicable calls of Appendix G.3 the two reach 0.801 and 0.92.

G.3 WHAT THE REASONING SAYS

The model-based annotation labels none of the 712 events as stating an explicit interface rationale. The 712 events are the applicable calls of Table 19 in four groups: *read* and *search*, each through the shell and through an API tool. For every event gpt-oss-20b scored its own reasoning text by forced-choice log-probabilities on yes-or-no questions. They ask whether the text says that a tool or command failed or behaved unexpectedly, whether it says that the exact characters of the file matter, whether it indicates that the writer is about to modify a file, and whether it gives an explicit reason for obtaining the information in one particular way rather than another. The last label is positive in 0% of events in every group, while at least one of the other labels is positive in 40–58% of events, depending on the group.

G.4 UNREGISTERED NAMES

TOOLS reaches for names its harness never registered far more often than the other harnesses whose requests carry a menu (Table 20). On the trajectory set 2.01% of its calls use an unregistered name, against 0.04% in SEED, 0.86% in EVOLVED and 0.23% in CODEX. The rate of TOOLS is thus 54, 2.3 and 8.8 times theirs. HARMONYAGENT and AGENTMOE send no tool field in their requests, so the criterion does not apply to them. Among replayed calls the rate of TOOLS is 4.31%, 59 of 1 370.

Table 20: **Calls to names the harness never registered**, on the trajectory set of Table 3, each call counted once. Tasks are those whose instance id a recorded request of the harness names, so they are fewer than the 36 tasks of the merged set for the seed family and the 50 of CODEX.

Harness	Tasks	Calls	Unregistered	Rate (%)	Tasks affected
SEED	27	2 702	1	0.04	1
EVOLVED	29	2 562	22	0.86	3
TOOLS	25	2 239	45	2.01	12
CODEX	47	1 743	4	0.23	1

The unregistered names are near-misses of the evolved tools of TOOLS, mostly written with named parameters rather than the positional array of the menu. On the trajectory set 95.6% of them are

closest, as strings, to one of the evolved tools. In the argument pass the names are search_file (27 calls), search (14), open_file (13), search_files (3), run_file (1), find_file (1) and an empty name (1). Of these 60 calls, 42 pass named parameters, the convention of the repo_browser tools of HARMONYAGENT. Another 11 pass the positional args array of the tools of TOOLS, 6 a command string and 1 something else. All 535 calls to registered evolved tools use the args array.

After an unregistered name the next call rarely repeats it. Of the 45 failures on the trajectory set, the next call uses the registered name 15 times (33%), falls back to the shell 14 times (31%), switches to another registered tool 10 times (22%) and retries the same name 5 times (11%). The remaining failure ended its trajectory, so these shares sum to 97%.

G.5 THE STATE AND THE NAME

At this layer and position, the tested decoder does not establish reliable prediction of whether a name that reaches for an evolved tool is registered. Among the 594 calls that reach for an evolved tool, 59 use an unregistered name. A decoder on the state separates them with an AUC of 0.592 against a permutation null of 0.504 ± 0.060, which is 1.5 null SDs above its mean. Over all 1 370 replayed calls the same decoder reaches 0.716 against a null of 0.503, because only a call that reaches for an API tool can name a missing one and the state already carries the interface.

H THE TWO REWRITES AND THE PUSHES IN THE STATE

Rewriting past reads between shell and API forms shifts the API share by +40.8 and −36.2 points and the operation by 8.6 and 2.5, rewriting what the last search returned moves mainly the operation, and pushing the state along the frozen decoder's own direction changes the decoder readout without establishing targeted control of the next operation.

H.1 DESIGN AND THE FULL RUN OF THE FORM REWRITE

Each rewrite changes one part of a real TOOLS context and leaves every other byte identical, and each context is continued several times in both versions. The form rewrite switches every earlier range read in the context between a shell read with sed -n and the equivalent read_file call, in both directions, while the content each call returned stays byte-identical. A context enters with at least two such reads, all of which are rewritten, and a read_file call without a line range becomes cat. Because this content is kept, a rewritten call is followed by the output its original interface produced. The result rewrite switches what the last search returned between its real list of hits and {"content": "no matches"}, in both directions, while the call itself stays unchanged. The replacement is the reply grep_search gives when nothing matches, and when the last search went through the shell this reply replaces its output. The reverse direction borrows hits from another search on the same task.

The full run of the form rewrite takes 60 contexts in each direction and continues each 6 times per version. The full run of the result rewrite takes 90 contexts from hits to no matches and 19 in reverse and continues each 8 times per version. Both use a budget of 256 tokens. This budget cuts off calls revealed late, many edits among them (Appendix E.3), so the operation half of each rewrite rests mostly on searches and reads. Continuations are sampled from the full distribution at temperature 1.0 without top-p truncation, in both versions alike, while the seed family itself samples at 0.7 with a top-p of 0.95 (Appendix A.1).

The two halves are measured on the same classifiable calls, so their shifts share one scale. The operation shift is the total variation between the operation distributions of the two versions, and the interface shift is the change in API share on the same calls. A continuation without a tool call issues no call at all. A continuation without a classifiable call issues none, issues one whose operation the classifier cannot assign, or calls a name the harness never registered, and it enters neither shift. Intervals come from a bootstrap over contexts, and each shift is compared with a null that swaps the two versions within contexts. A total variation is positive even when nothing changes, so for the operation shift the null and not zero marks no change. Each context is also its own control: we count the contexts whose API share moves in the pooled direction among those that move at all and apply an exact sign test.

The frozen decoder reads the state at the first generated-token position and on the output of block 20, the position and the block it was fitted on. In every version all continuations share this state, because each began with the same token, so the decoder is read once per version of each context, whether or not a continuation issues a call. It was fitted once on the 8 172 classified calls of the six harnesses, which can include the call recorded after an original context but never one made after a rewritten context. Its shift is the total variation between the mean posteriors of the two versions, compared with the same within-context null.

Table 21: **The form rewrite in full** (rows 10–11 of Table 1). Shifts and shares in points or %, brackets 95% context-bootstrap intervals, null_{95} the 95th percentile of the within-context swap null and p its permutation p. The operation and interface rows use the same classifiable calls, the state rows one state per version of each context, read on the output of block 20. A shift in a share is the rewritten version minus the original.

	shell → API	API → shell
Contexts	60	60
Continuations per version, all contexts	360	360
Classifiable calls, original and rewritten	239 and 204	245 and 275
Operation		
Operation shift (TV)	8.6 [3.2, 16.3]	2.5 [2.1, 9.8]
null_{95} and p	8.4, $p = 0.046$	7.8, $p = 0.79$
Shift in the *search* share	−8.4	−0.2
Shift in the *read* share	+6.0	−0.1
contexts moving with it, sign test	19 of 27, $p = 0.052$	17 of 27, $p = 0.25$
Interface		
API share, original → rewritten (%)	3.8 → 44.6	64.9 → 28.7
Interface shift (Δ API share)	+40.8 [+30.4, +51.2]	−36.2 [−46.7, −25.9]
null_{95} of $\lvert\Delta\rvert$ and p	15.5, $p = 1.0 \times 10^{-4}$	13.9, $p = 1.0 \times 10^{-4}$
contexts moving with it, sign test	33 of 34, $p = 4.1 \times 10^{-9}$	39 of 41, $p = 7.8 \times 10^{-10}$
|Interface shift| over operation shift	4.7	14.3
State at the first generated-token position		
Frozen decoder, TV of mean posteriors	2.2 [0.9, 4.0]	0.8 [0.4, 2.3]
null_{95} and p	2.1, $p = 0.031$	1.7, $p = 0.51$
Decoder shift in $P(\text{read})$	+1.9 [+0.4, +3.5], $p = 0.023$	+0.3 [−1.0, +1.6], $p = 0.70$
Decoder shift in $P(\text{search})$	−1.4	+0.3
Named axes passing Holm, of 16	0	0
The context		
No tool call, original → rewritten (%)	33 → 33	17 → 19
Unregistered-name calls, original → rewritten	0 → 34	54 → 15
contexts with one, original and rewritten	0 and 25	25 and 13

The form rewrite moves the interface in both directions and the operation far less (Table 21). Turning shell reads into read_file calls raises the API share of the next calls from 3.8% to 44.6%, and the reverse rewrite lowers it from 64.9% to 28.7%. The share moves the predicted way in 33 of 34 and 39 of 41 contexts, counting those in which it moves at all. From shell to API the operation shift of 8.6 points only just clears its null of 8.4 ($p = 0.046$), as some searches give way to reads, and the *read* share rises in 19 of 27 contexts that move ($p = 0.052$). From API to shell the operation shift of 2.5 points stays inside its null ($p = 0.79$). The observed ratios of the interface shift to the operation shift are 4.7× and 14.3×.

Under the form rewrite the frozen decoder moves little, and beyond its null only from shell to API, where the operation moves as well. In that direction it shifts by 2.2 points against a null of 2.1 ($p = 0.031$), and its $P(\text{read})$ changes by +1.9 points, in the direction of the *read* share (+6.0). From API to shell it shifts by 0.8 points ($p = 0.51$), although the API share moves by at least 36.2 points in each direction. No named axis passes Holm in either direction.

The no-tool-call rate stays similar under the rewrite, while the calls to unregistered names change substantially. Continuations without a tool call are 33% before and 33% after the rewrite from shell to API, and 17% and 19% from API to shell. Rewriting reads into read_file calls raises the calls to

unregistered names from 0 to 34, and the reverse rewrite lowers them from 54 to 15. These calls enter neither shift.

H.2 THE FULL RUN OF THE RESULT REWRITE

Table 22: **The result rewrite in full** (rows 12–13 of Table 1), measured as in Table 21. The difference of the two share changes is taken per context over all continuations, including those without a call, and tested with a Wilcoxon test over the contexts in which it is not zero.

	hits → no matches	no matches → hits
Contexts	90	19
Continuations per version, all contexts	720	152
Classifiable calls, original and rewritten	570 and 575	70 and 63
Operation		
Operation shift (TV)	29.0 [20.7, 38.0]	12.1 [6.6, 22.0]
null_{95} and p	12.0, $p = 1.0 \times 10^{-4}$	13.3, $p = 0.10$
Shift in the *search* share	+24.0	−4.3
Shift in the *read* share	−29.0	+12.1
contexts moving with it, sign test	45 of 59, $p = 6.5 \times 10^{-5}$	8 of 9, $p = 0.039$
Search minus *read*, all continuations	+42.2	−9.9
contexts moving with it, Wilcoxon	53 of 72, $p = 7.2 \times 10^{-7}$	9 of 13, $p = 0.12$
Interface		
API share, original → rewritten (%)	24.2 → 17.7	72.9 → 65.1
Interface shift (Δ API share)	−6.5 [−12.7, −0.9]	−7.8 [−26.4, +8.3]
null_{95} of $\lvert\Delta\rvert$ and p	6.1, $p = 0.035$	15.9, $p = 0.50$
contexts moving with it, sign test	23 of 34, $p = 0.058$	6 of 9, $p = 0.51$
Operation shift over |interface shift|	4.5	—
State at the first generated-token position		
Frozen decoder, TV of mean posteriors	18.1 [12.5, 24.2]	7.6 [4.3, 12.7]
null_{95} and p	7.2, $p = 1.0 \times 10^{-4}$	7.1, $p = 0.029$
Decoder shift in $P(\text{read})$	−18.0 [−24.0, −12.3]	+2.2 [−2.6, +7.4]
p	1.0×10^{-4}	0.44
Decoder shift in $P(\text{search})$	+12.4	+5.4
Named axes passing Holm, of 16	8	2
The context		
No tool call, original → rewritten (%)	16 → 16	43 → 49
Unregistered-name calls, original → rewritten	35 → 23	15 → 14
contexts with one, original and rewritten	18 and 12	8 and 7

The result rewrite moves mainly the operation (Table 22). When a hit list becomes no matches, the operation shifts by 29.0 points against a null of 12.0 ($p = 1.0 \times 10^{-4}$). The *read* share of the next calls changes by −29.0 points and the *search* share by +24.0, and the *read* share falls in 45 of 59 contexts that move ($p = 6.5 \times 10^{-5}$). Taken per context over all continuations, the change in the *search* share minus the change in the *read* share is +42.2 points (53 of 72 non-tied contexts, Wilcoxon $p = 7.2 \times 10^{-7}$).

The interface moves far less. The API share falls from 24.2% to 17.7%, a shift of −6.5 points that clears its null of 6.1 ($p = 0.035$). It falls in 23 of 34 contexts that move, which a sign test does not separate from chance ($p = 0.058$). The operation shift is 4.5 times the interface shift.

Putting hits back after a search that found nothing moves neither half beyond its null. On 70 and 63 classifiable calls the operation shifts by 12.1 points ($p = 0.10$), mostly as the *read* share rises by +12.1 points, and the API share moves by −7.8 ($p = 0.50$). In this direction 43% of continuations issue no tool call before the rewrite and 49% after it, against 16% in the forward direction, so it rests on fewer calls.

The frozen decoder moves most where the operation moves most. From hits to no matches its mean posterior shifts by 18.1 points against a null of 7.2 ($p = 1.0 \times 10^{-4}$). Its $P(\text{read})$ changes by −18.0 points and its $P(\text{search})$ by +12.4, in the direction of the *read* and *search* shares. From no matches

to hits it shifts by 7.6 points, just beyond its null of 7.1 ($p = 0.029$), while the operation stays inside its own. In that direction its $P(\text{search})$ changes by +5.4 points while the *search* share changes by −4.3, so the posterior does not follow the calls there. Across the four directions of the two rewrites the decoder shifts of 18.1, 7.6, 2.2 and 0.8 points fall in the same order as the operation shifts of 29.0, 12.1, 8.6 and 2.5 points. Holm passes 8 of the 16 named axes from hits to no matches and 2 in reverse.

H.3 Earlier runs

Two earlier runs of the form rewrite reproduce the interface result. The first continued 60 contexts for 480 generations but stored only aggregate counts, so its tests are unpaired. It moved the API share by +40.6 points from shell to API (Fisher $p = 1.4 \times 10^{-8}$) and by −24.3 points from API to shell ($p = 5.7 \times 10^{-4}$), with operation shifts of 4.7 and 5.7 points. The second continued 30 contexts per direction for 1 440 generations and is measured as the full run. From shell to API the three runs give interface shifts of +40.6, +40.9 and +40.8 points and operation shifts of 4.7, 4.6 and 8.6 points, and from API to shell the second run and the full run agree on both halves.

An earlier run of the result rewrite, on 49 contexts for 392 generations, agrees with the full run from hits to no matches. On its 91 classifiable calls the *read* share fell from 67.0% to 40.7% and the *search* share rose from 31.9% to 58.2%, while the API share moved from 20.9% to 13.2%, an operation shift 3.6 times the interface shift.

H.4 Pushes in the state

Pushing the state along the frozen decoder's own direction drives the decoder readout to near certainty without establishing targeted control of the next operation. No push along the direction differs from a random push of the same norm, and increasing the push toward *search* degrades output validity (Table 23, Figure 9). On the four- and eight-SD conditions the decision rule returns *not established*.

Design. The contexts are the first 40 contexts of the result rewrite from hits to no matches, in their original version, so the last search in each returned real hits and *read* is the most common next operation. Only the state is changed. The direction is the one along which the frozen decoder's logit of *search* minus its logit of *read* grows fastest in the 64 coordinates of the frame. It is mapped into the residual stream through the frame, so a push moves these coordinates along the direction and along no other. The push is added to the output of block 20, the state on which the frame and the decoder were fitted, at 24 consecutive positions from the last prompt token on. The size α of a push counts token-level SDs of the state's coordinate along the direction, measured on the generated tokens of TOOLS, so one SD is a push of norm 166.8, and a positive α pushes toward *search*. A push of two, four or eight SDs has about 2.5, 5.0 or 9.8% of the norm of the state. The control is a random direction in the residual stream, scaled to the same norm and drawn anew for every context, sample and size. Continuations are sampled as in the rewrites, from the full distribution at temperature 1.0 without top-p truncation and with a budget of 256 tokens. Each context is continued three times per condition with seeds shared across conditions, so the conditions are paired by context and sample. The main run pushes by four and eight SDs in each direction (1 080 continuations). A second run by one and two SDs (1 080 continuations) locates where damage to the output begins, and the continuations without a push are identical in the two runs.

Measurement. Each push is compared with no push as a rewritten context is compared with its original (Appendix H.1): operation shift and change in API share on the classifiable calls, context-bootstrap intervals and a null that swaps the two conditions within contexts, each with 10 000 draws. The decoder is read after the push at the first generated-token position. The rule calls the direction effective only if four conditions hold at one size of push. (i) The push along the direction differs from the random push of the same size in the operation it pushes toward, with a two-sided Fisher $p < 0.05$. (ii) The difference points the same way on well-formed continuations alone. (iii) Pushes toward *search* and toward *read* move the operation in opposite directions. (iv) The well-formed share under the push stays within ten points of the random push. Condition (i) without (iv) counts as damage to generation.

Table 23: **Pushes along the frozen decoder's direction and along a random direction of the same norm**, 40 contexts and 120 continuations per condition. α in token-level SDs, negative toward *read*. *Calls*: classifiable calls. *WF*: well-formed continuations (%). *Decoder posterior*: mean $P(\text{read})$ and $P(\text{search})$ after the push at the first generated-token position (%) and the TV from no push (points). Every decoder shift along the direction lies beyond its null ($p = 1.0 \times 10^{-4}$) and none of the random ones does. *Next call*: *read* and *search* shares of the classifiable calls (%), operation shift from no push with null_{95} and within-context swap p, and change in API share (points) with its p. Without a push the API share is 21.9%. *Fisher*: the push along the direction against the random push of the same α, in the share of the operation pushed toward. The pushes of two and four SDs along the direction are rows 14–17 of Table 1.

	Output		Decoder posterior			Next call							
α	Calls	WF	*read*	*search*	TV	*read*	*search*	TV	null_{95}	p	ΔAPI	p	Fisher p
none	96	86	43.9	36.3	—	59.4	28.1	—	—	—	—	—	—
Along the decoder's direction													
−8	81	82	100.0	0.0	56.1	55.6	30.9	3.9	12.6	0.87	−2.1	0.60	1.00
−4	97	96	99.2	0.0	55.3	54.6	33.0	6.9	12.1	0.36	+0.8	0.87	0.77
−2	88	92	91.3	2.6	47.4	56.8	35.2	7.6	10.9	0.30	−3.7	0.36	0.77
−1	91	91	73.3	12.9	29.4	57.1	31.9	5.2	10.2	0.57	+1.2	0.78	0.77
+1	86	85	17.0	64.9	29.1	61.6	32.6	6.7	11.2	0.41	−6.8	0.013	1.00
+2	84	74	3.6	84.5	48.2	51.2	32.1	9.2	10.7	0.12	−0.4	0.89	0.53
+4	21	64	0.1	96.3	60.0	71.4	23.8	12.1	23.7	0.50	+49.6	1.0×10^{-4}	0.60
+8	0	0	0.0	99.8	63.5	—	—	—	—	—	—	—	—
Along a random direction of the same norm													
−8	93	92	43.6	36.6	0.4	54.8	32.3	7.5	10.7	0.29	−2.5	0.34	
−4	90	88	43.5	36.8	0.8	57.8	31.1	5.6	9.0	0.35	−0.8	0.89	
−2	95	87	43.7	36.1	0.4	53.7	35.8	10.8	9.9	0.023	−0.8	0.90	
−1	96	88	43.8	36.4	0.2	59.4	34.4	6.2	10.3	0.34	−3.1	0.53	
+1	101	88	44.0	36.3	0.2	58.4	33.7	5.5	9.0	0.37	−2.1	0.64	
+2	94	88	43.8	36.0	0.3	57.4	37.2	9.1	9.0	0.048	−2.7	0.50	
+4	92	90	44.0	36.0	0.5	58.7	32.6	5.6	9.8	0.45	−4.5	0.17	
+8	91	88	43.1	36.1	1.5	54.9	33.0	6.3	11.4	0.41	+2.3	0.58	

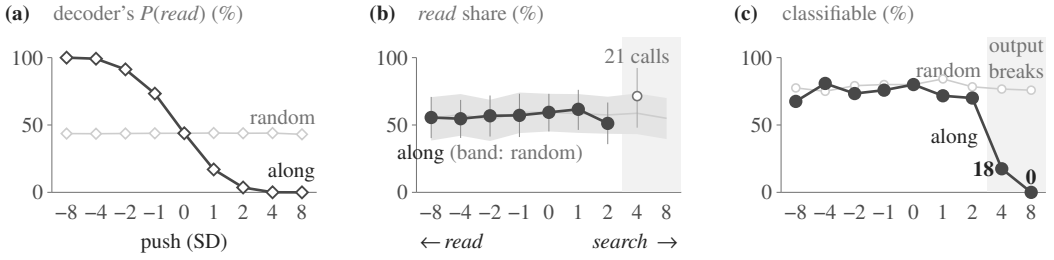


Figure 9: **Dose-response of the state push.** Push along the frozen decoder's direction (dark) and along a random direction of the same norm (grey), α in token-level SDs, negative toward *read*, over 40 contexts; shaded: $\alpha \geq +4$. (a) Decoder's mean $P(\text{read})$ at the first generated-token position. (b) *read* share of the classifiable next calls (whiskers and band: 95% context bootstrap; hollow: under 50 classifiable calls; none at $\alpha = +8$). (c) Share of continuations that issue a classifiable call.

Results. The push moves the decoder readout along its direction, and a random push of the same norm does not (Figure 9a). Pushes of two and four SDs toward *read* change the decoder's mean $P(\text{read})$ by +47.4 and +55.3 points, from 43.9% without a push to 99.2% at four SDs. Four SDs toward *search* change $P(\text{search})$ by +60.0 points, to 96.3%. No random push moves the posterior by more than 1.5 points.

Toward *read* the next operation does not follow, and up to four SDs the output stays well formed (Figure 9b). The operation shift stays inside its null at every size and reaches at most 7.6 points,

against a null of 10.9. At four SDs the *read* share of the next calls is 54.6%, against 59.4% without a push, and 96% of continuations are well formed, against 86%.

Increasing the push toward *search* degrades output validity without establishing a targeted increase in search calls (Figure 9c). At two SDs the operation shift of 9.2 points stays inside its null (p = 0.12), while the well-formed share falls to 74%. At four SDs only 21 of 120 continuations issue a classifiable call, and 23.8% of these are searches, against 28.1% without a push. At eight SDs no continuation is well formed. On the few calls left at four SDs the API share changes by +49.6 points ($p = 1.0 \times 10^{-4}$). Most of these continuations run to the token budget, and shell commands break before API calls do, so 71.4% of the calls left are API calls. We read this rise as a selection by the broken output and not as a push on the interface.

The decision uses the four- and eight-SD conditions. In none of those with classifiable calls does a push along the direction differ from the random push of the same size in the share of the operation it pushes toward (Figure 9b), so condition (i) fails and the rule returns *not established*. The one- and two-SD runs agree. Over all seven direction-versus-random comparisons the Fisher tests give $p \geq 0.53$, and p = 1.00 after a Holm correction over the seven. Three further tests of the next call fall below p = 0.05: the random pushes of two SDs move the operation just beyond its null (p = 0.023 and p = 0.048), and the push of one SD toward *search* along the direction moves the API share by −6.8 points (p = 0.013). None survives a Holm correction over the push conditions.

A random direction among the 2880 dimensions of the residual stream barely moves the 64 coordinates of the frame, so this comparison tests whether the decoder's direction adds to a push of the same norm.

H.5 Further experiments

The results suggest several further experiments.

Crossed rewrites. Applying the form rewrite and the result rewrite to the same contexts, alone and together, would test whether the two halves move independently within one context. The present rewrites use different sets of contexts.

A stricter control for the push. A random direction drawn within the coordinates of the frame and scaled to the same norm would move the frame as far as the decoder's direction does. Comparing the two would separate an effect of the decoder's direction from an effect of moving the frame at all.

Common snapshots. Restoring one file-system snapshot and continuing both harnesses from it would replace the matched anchors of Section 6 with a comparison that shares its entire history.

Naming repair. Registering the names that the model produces, such as search_file and open_file, as aliases and accepting named parameters would test whether the unregistered-name failures of Section 7 can be repaired from the harness side, measured by the rate of unregistered names and by what the next call does.

Selection with readouts. An improvement procedure could use the readout in its offline stage. It would collect the applicable calls of a target operation with their readouts and, for each candidate change, sample short continuations from the same snapshots. It would score whether a reasonable operation is still chosen, whether it is written through a registered interface and whether the needed content is obtained, and then validate the surviving candidate on held-out tasks by resolve rate, correctness and cost. The comparison that matters is with selection from text and behaviour alone under the same candidates and budget, with a control in which the readouts are shuffled across snapshots. Only if real readouts beat shuffled ones does the readout improve the choice of what to change.