WHEN DOES ACTIVATION STEERING CHANGE WHAT A MODEL COMPUTES FROM?

Anonymous authors
Paper under double-blind review

ABSTRACT

Activation steering can reliably change an agent's output by modifying its internal activations. Yet arriving at the same answer need not involve the same computation: behavioral equivalence does not imply mechanistic equivalence. We test whether an activation edit changes the state used in subsequent computation or biases that computation toward the desired output. In a controlled state-tracking task, trace supervision creates an editable register: changing its internal value makes the model apply the next operation to the edited state. We next ask whether widely used mean activation steering admits a similar interpretation: does steering recreate the internal configuration the model naturally uses when performing a task? If so, transplanting that activation should be effective, and steering should remain effective at the scale of the natural source-to-target activation change. Neither prediction holds in the two selected model–task settings. Replacing the activation at one layer produces less than 2% of the target-answer margin gain from patching through all remaining layers, while natural-scale steering is similarly ineffective. The steering vectors have norms 77 and 26 times the median natural change in Qwen and Llama, producing 54% and 82% of the reference effect. Thus, a successful steering intervention need not reproduce the natural target activation at the intervention layer. More generally, an intervention should be interpreted as changing the computational state only when a later computation uses the edited value according to the semantics of that state.

1 INTRODUCTION

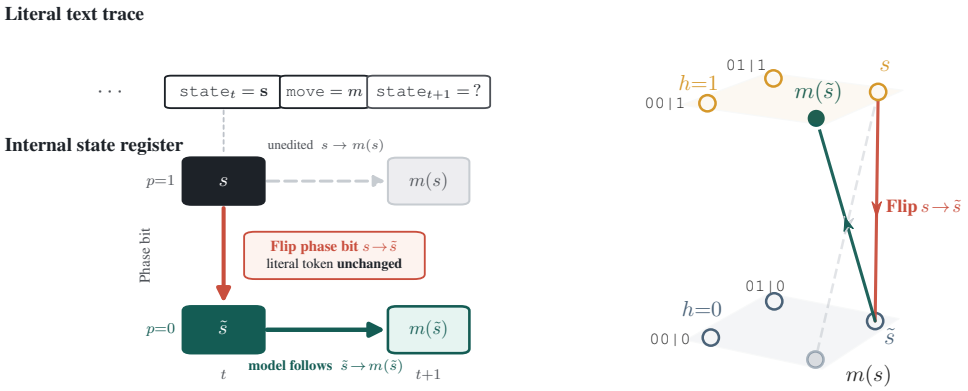
Activation steering changes the behavior of an agent by adding a direction to its internal activations (Turner et al., 2023; Todd et al., 2024; Zou et al., 2023). The direction is often constructed by averaging activations associated with a task or behavior. Successful steering may suggest that the intervention writes a task state the model subsequently uses in computation. On the other hand, it could also simply be biasing later computation towards the same output. Distinguishing these explanations requires testing how subsequent computation uses the edited activation.

We fine-tune multiple 7B language models to write the running state of a known transition system. At evaluation, we change the internal representation of one written state while leaving the visible scratchpad unchanged. Because the update rule is known, a genuine state change has a unique downstream consequence: the model should apply the next operation to the edited state rather than the original one. The resulting continuation follows the edited state and changes appropriately when the next operation changes, showing that the model is computing from the edited value rather than simply producing a different answer.

However, this result does not imply that every behavior-changing edit works this way. We next test the *correspondence hypothesis (CH)* that Function Vectors (Todd et al., 2024), a widely used form of mean activation steering, recreate the activation the model naturally produces at the intervention layer when performing the target task. If CH holds, the target-run activation should be effective when transplanted into the source run, and the steering vector should remain effective when scaled to the natural activation difference between the runs.

We find that neither prediction holds in the two tested settings. These results separate three properties often conflated in steering work: behavioral efficacy, directional specificity, and state like use. A steering vector can exhibit the first two without establishing the third.

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.



(a) The editing mechanism

(b) A sample divergence from the edit

Figure 1: **Editing the register redirects the computation.** The visible trace token reports state (s), but a low-rank internal edit changes its hidden state to $\tilde{s}$. Under the same upcoming move m, the model computes $m(\tilde{s})$ rather than $m(s)$. The outcome therefore depends on both the edited state and the operation applied after the edit.

Our contributions are as follows:

1. **A causal test for computational state.** We distinguish changing an answer, changing later computation, and changing a state variable by asking what future operations the edited activation can support.
2. **A positive state-editing example.** Counterfactual register editing shows that trace supervision can make an explicit scratchpad token into internal state that the model reads and updates. Controls separating state from answer replicate in a second 7B model family.
3. **A natural-activation test for mean steering.** In Qwen and Llama, a large steering vector changes behavior while the matched target activation at one layer and a natural-sized steering edit do not. Steering success therefore does not by itself establish that a natural target state has been transplanted.
4. **A model-dependent specificity result.** Llama responds selectively to the task's steering direction, while Qwen responds to several structured directions of the same size. The specificity of a steering vector must be measured rather than assumed from its construction.

2 Related Work

Activation steering methods, including Activation Addition, representation engineering, inference-time intervention, and Function Vectors, show that internal directions can reliably control language-model behavior (Turner et al., 2023; Zou et al., 2023; Li et al., 2023b; Todd et al., 2024). Because behaviorally equivalent interventions need not identify a unique internal representation (Venkatesh and Mahendran Kurapath, 2026), we ask whether a successful steering vector recreates an activation that the model naturally uses. This connects to causal abstraction and interchange interventions (Geiger et al., 2024; 2021) and to reusable-interface methods such as Hidden APIs (Ma et al., 2026). Our controlled register setting makes the state and transition rule known, allowing each edit to be tested through its operation-specific downstream consequence. A fuller discussion of activation patching, scratchpads, state tracking, world models, and representation geometry appears in Appendix G.3.

3 A Controlled Scratchpad State-Tracking Task

The model reads a sequence of moves and reports the resulting state, emitting the state after each move as a short running trace before the final answer. Each state has two parts: a *visible coordinate*

that can be read directly from the moves, and a hidden coordinate that depends on the *order* in which the moves were applied and that the visible coordinate alone does not determine. We write the task as a walk on a finite group, which makes the hidden coordinate precise and gives us ground truth for every intervention.

Formally, a prompt is a sequence of moves $m_1, \ldots, m_T$ drawn from a finite group G acting on itself by left multiplication. The states satisfy

$$s_0 = e, \qquad s_t = m_t\, s_{t-1}, \quad t = 1, \ldots, T.$$

The model reports s_T; running-state supervision additionally asks for each intermediate s_t. Each state has a *visible coordinate* $v(s)$ and a *hidden bit* $h(s)$:

$$s \longleftrightarrow \big(v(s), h(s)\big), \qquad v(s) \in V = \mathbb{Z}_2 \times \mathbb{Z}_2, \quad h(s) \in \mathbb{Z}_2.$$

The visible coordinate is a running tally of the moves that ignores their order, so it can be read straight off the sequence; the hidden bit instead depends on the *order* in which the moves were applied, so it can only be obtained by tracking the sequence as it is consumed. We use the two non-commutative groups of order eight, the quaternion group Q_8 and the dihedral group D_8. Both have the same visible quotient V, the Klein four-group, but differ in how their multiplication updates the hidden coordinate. The group-theoretic construction is deferred to Appendix A.1; Figure 6 gives a worked example prompt.

Why this setting. Prior work has studied representations of cyclic states, such as days and months (Engels et al., 2024; Kantamneni and Tegmark, 2025; Feucht et al., 2026), and how chain-of-thought can emulate finite automata (Zhang et al., 2025). Here, changing the hidden bit while holding the visible coordinate fixed gives a counterfactual state whose continuation is determined by the next move. This lets us test whether the model updates the edited state according to the task rule.

Unless otherwise stated, we analyze a `Qwen2.5-Coder-7B` model trained with full running-state supervision; auxiliary supervision arms are reported in Appendix B.1.

4 Counterfactual register editing

To test whether the intermediate token carries an executable state, we edit a low-rank *register* at the current-state token and ask whether the model recomputes its next state from the edited value. The method has three parts: identify a low-rank state register, edit it to a counterfactual value, and score whether the next-step computation follows the edited state (Figure 1 and Figure 6).

Intervention. Let $z_{\ell,i}(x) \in \mathbb{R}^d$ denote the residual stream at layer ℓ and token position i on prompt x, and let $i^\star$ be the position of the current-state token. We estimate the register subspace $U \subseteq \mathbb{R}^d$ at $(\ell, i^\star)$ from the class-conditional means of the residual stream: for each visible value we take the difference of means between the two hidden values, and U is the span of the leading $r = 16$ such directions at the validation-selected layer 12 edit site. The subspace U and the class means $\mu(s)$ are estimated on a calibration split disjoint from all items used to report CUS and register agreement, and the layer and site are locked on a validation split before the causal test. Writing Π_U for the orthogonal projector onto U and $\mu(s)$ for the mean residual stream in state s, the counterfactual state flips only the hidden coordinate, and the edit adds the projected class-conditional mean difference:

$$\tilde{s} = (v, 1 - h), \quad \tilde{z}_{\ell,i^\star} = z_{\ell,i^\star} + \Pi_U\big(\mu(\tilde{s}) - \mu(s)\big).$$

The decoded hidden coordinate reads $h(\tilde{s})$ while the visible coordinate and the printed token are unchanged. The *random-subspace* and *orthogonal-complement* controls apply the same-rank edit in a random subspace or within $U^\perp$; the *full-residual* control replaces the entire token, $\tilde{z}_{\ell,i^\star} = \mu(\tilde{s})$. To check that the hidden coordinate exists at all we also fit a linear readout of it from the same token and report its held-out accuracy, which distinguishes a decodable variable from a causally used one (Geiger et al., 2024; Turner et al., 2023).

We provide the correct sequence of intermediate states up to the intervention, then intervene on the residual stream at the current hidden-bit token, leaving the literal token unchanged, and measure the model's prediction of the next hidden bit. Appendix B.3 separately tests whether the register is still used when the model generates the trace itself.

Evaluation. Let $\hat{h}_{t+1}$ be the model's predicted hidden coordinate of the next state under the edit, given the upcoming move m. If the model executes its learned update on whatever state is written into the register, then $\hat{h}_{t+1}$ should equal $h(m \cdot \tilde{s})$, the hidden bit of the move applied to the *edited* state, rather than $h(m \cdot s)$, the move applied to the *original* state. On the discriminating subset $\mathcal{D} = \{\, h(m \cdot \tilde{s}) \neq h(m \cdot s) \,\}$ we define *register agreement* and *Counterfactual Update Selectivity*,

$$R \;=\; \mathbb{P}\big(\hat{h}_{t+1} = h(m \cdot \tilde{s})\big), \qquad \text{CUS} \;=\; \mathbb{P}\big(\hat{h}_{t+1} = h(m \cdot \tilde{s})\big) - \mathbb{P}\big(\hat{h}_{t+1} = h(m \cdot s)\big).$$

R measures agreement with the counterfactual next state while CUS measures preference over the original next state.

Two controls separate supplying a state from supplying an answer. The *move-swap* selectivity varies the move on a fixed edited state, evaluated on items where $h(m \cdot \tilde{s}) \neq h(m' \cdot \tilde{s})$:

$$T = \mathbb{P}\big(\hat{h}_{t+1} = h(m \cdot \tilde{s})\big) - \mathbb{P}\big(\hat{h}_{t+1} = h(m' \cdot \tilde{s})\big).$$

The *conflicting-continuation* selectivity injects a real occurrence $\tilde{s}_{\text{src}}$ of the counterfactual state whose own continuation h_{src} conflicts with the move-applied target, $h(m \cdot \tilde{s}) \neq h_{\text{src}}$:

$$L = \mathbb{P}\big(\hat{h}_{t+1} = h(m \cdot \tilde{s})\big) - \mathbb{P}\big(\hat{h}_{t+1} = h_{\text{src}}\big).$$

Localization. We localize the update by measuring how ablation, matched-control, and restoration interventions affect CUS (Conmy et al., 2023; Syed et al., 2023; Meng et al., 2022; Todd et al., 2024). Further, we test a fixed set of attention edges and contiguous groups of low rank components on the path to the next-state readout. An ablation masks a candidate component c during the forward pass and recomputes CUS_c; the *relative reduction* is $\kappa_c = (\text{CUS}_\varnothing - \text{CUS}_c)/\text{CUS}_\varnothing$, where $\text{CUS}_\varnothing$ is the unintervened baseline. A matched control routes c from the visible coordinate rather than the hidden coordinate, and a restoration recovers c's contribution from the clean edited run. Table 6 lists the candidate components and Appendix C.1 provides the predefined localization criteria.

5 Trace supervision creates an executable state register

We organize the evidence as an edit-site check and three causal tests, each ruling out a specific alternative explanation:

1. **Edit-site check**: is the literal hidden-bit token even readable? It is the literal trace token, so it decodes in every arm and does not distinguish them: the causal evidence begins with *use*.
2. **Use**: does editing that coordinate change what the model computes next? A rank-16 edit redirects the next state (Section 5.1).
3. **Move-specific**: does the model apply the *actual* upcoming move to the edited state, rather than emit a fixed answer? Swapping the move swaps the next state in step (Section 5.1).
4. **Conflicting continuation**: does the model recompute its next state from the edit, or copy a pasted-in continuation? When the injected state carries a conflicting continuation, the model follows the current move and not that continuation (Section 5.1).

Figure 2 summarizes these matched tests, and Table 3 reports their values.

The current state is perfectly decodable as it is written into the trace. On 600 held-out items, late-layer decoding accuracy for the next-state coordinate exceeds 0.96 in both Q_8 and D_8 as exhibited in Figure 7.

5.1 Editing the register redirects the computation

On held-out items, a rank-16 same-position edit makes the running-state model apply the upcoming move to the counterfactual state: next-state agreement is 0.80 in Q_8 and 0.91 in D_8, versus about 0.02 for equal-rank random and orthogonal-complement edits. Full-residual replacement is no stronger, so the compact subspace captures the causal effect available at the site. Because the edit is made at the current-state token, before the next-state computation and away from the answer, it redirects an intermediate computation rather than simply pushing the output.

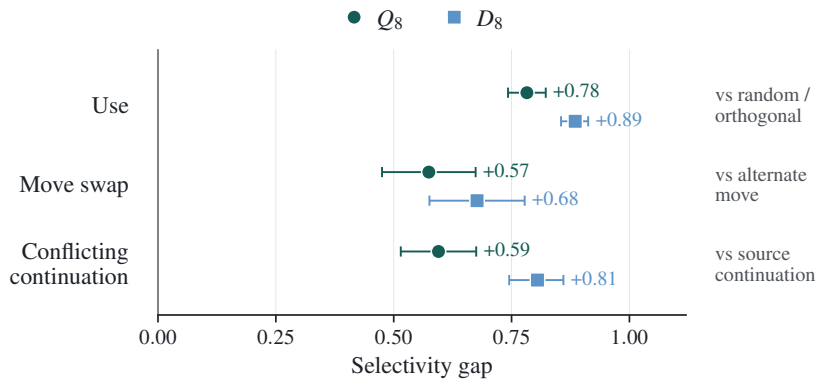


Figure 2: **Summary of the causal chain.** The three rows report complementary selectivity gaps testing whether the edited coordinate is used as an executable state rather than merely being readable. Each row uses the matched alternative shown at right, so their magnitudes are not directly comparable. Teal circles denote Q_8, blue squares denote D_8, and whiskers are 95 percent bootstrap intervals on held-out items; all exclude zero.

Two controls separate supplying a state from supplying an answer. In the move-swap control, holding the edited state fixed and changing the upcoming move changes the next state in the corresponding way: the move-specific next state exceeds the alternative by 0.57 in Q_8 and 0.68 in D_8. In the conflicting-continuation control, we inject a single real occurrence of the counterfactual state whose own continuation conflicts with the move-applied target. The model follows the current move, $m \cdot \tilde{s}$, on 0.80 of items and the injected continuation on only 0.20, a selectivity of +0.595; the same control on D_8 gives +0.805, so the edit supplies a state and not an answer in both groups. A single injected occurrence is as effective as the constructed counterfactual, so the effect is not an artifact of averaging. Together these controls show the model runs its learned update on whatever state is written into the register, rather than emitting a fixed answer. In `Mistral-7B`, register agreement reaches 0.93 in Q_8 and 0.94 in D_8. Move-swap and conflicting-continuation selectivities range from 0.84 to 0.89, replicating the operation-dependent use of the edited state.

We also edit traces generated by the model, retaining only those that are correct through the edited token in order to test whether the redirection persists. Indeed, we find persistence through later states and the final answer: in D_8, the edited branch is followed on about 0.86 of items at each of the next three states, whereas Q_8 falls below the one-step criterion but retains above-chance multi-step persistence and positive move-specific and conflicting-continuation controls.

5.2 Localization

The register edit is low-rank in both groups, but the route that updates it differs. The counterfactual effect saturates by rank 1 in Q_8 and rank 2 in D_8, far below the 3584-dimensional residual stream. In Q_8, ablating a single layer-22 attention edge from the current-state hidden coordinate to the next-state readout removes 94% of CUS; matched controls preserve the effect, restoration recovers it, and the result replicates on an independent split. In D_8, no single component removes more than 24%. A frozen four-edge route removes 68.7% of CUSacross ten reported splits, eight of which are held-out confirmatory, supporting a partial distributed localization. Thus, a compact register need not have a similarly concentrated update route. Appendix B reports complete results; Appendices B.2–B.3 cover replication and generated traces, and Appendix C details localization.

6 State edits beyond group operations

We extend the assay using task-specific `Qwen3-8B-Base` adapters with additional training on original and visibly edited continuations. At evaluation, one rank-16 activation edit changes a

supplied state-value representation while leaving the written input unchanged. The model generates subsequent states, scored by an exact simulator.

A 32-state system combines location, mode, and counter. Under prompt-specified rules, the same mode or counter edit supports two future actions that move the location in opposite directions. Across eight held-out rule systems, both actions yield correct complete continuations on all 128 cases after four intervening updates, separately for two independently trained models. The future actions follow the edited token, so the edited activation cannot already encode which action will follow (Figure 12).

In a tiny interpreter, editing a register, flag, or stack value changes a later branch. With three unrelated instructions before first use, all 128 held-out programs take the correct branch and 127/128 complete continuations are exact. With seven such instructions, branch accuracy remains 126/128, but exact continuations fall to 71/128 (Figures 13 and 14).

In both tasks, overwriting the edited field before use restores the unedited later outcomes on all 128 cases. These results extend causal state use to interacting variables and delayed program decisions; protocols, comparisons, and limitations are in Appendix D.

7 Comparisons with Modern Steering Techniques

The preceding experiments identify a causal register whose value can be edited to redirect subsequent computation. We now ask whether existing activation-steering methods act through the same mechanism.

7.1 Function Vector Steering

We use Function Vectors (FVs) (Todd et al., 2024) to compare activation steering with counterfactual register editing. We first test whether FV steering writes into the register identified in the state-tracking model. We then examine whether effective steering in lexical tasks recreates the model's natural target activation.

To illustrate the construction, consider English-to-Spanish translation with ten input-output demonstrations followed by a query. For each query, we construct a correct prompt and a shuffled prompt. In the correct prompt, each input is paired with its correct output. In the shuffled prompt, the same inputs, the same set of outputs, and the same query are retained, but the outputs are reassigned across demonstrations using a fixed permutation. For example, *dog* $\rightarrow$ *perro* and *cat* $\rightarrow$ *gato* might become *dog* $\rightarrow$ *gato* and *cat* $\rightarrow$ *perro*. This breaks the task demonstrated by the examples while preserving the prompt content and final answer position, giving a controlled prompt on which steering can restore the target behavior.

The method identifies attention heads whose outputs most improve the target behavior when inserted into shuffled prompts. It averages the selected head outputs at the final prompt token over 45 correct prompts and projects the result back into the model's hidden representation. Following the original work, we call the resulting direction the FV. The details of how the FV was constructed for the Q_8 and D_8 task are in appendix E.1.

7.2 Function Vector steering does not directly write the identified registers

FV steering is effective by the usual behavioral criterion. At the intervention scale selected on the training set, it produces the target one-step next state on 65.6% of training examples and 64.3% of held-out Q_8 examples. This approaches the efficacy of counterfactual register editing, which achieves one-step register agreement of 80% and free-running final-answer agreement of 72%.

Judged only by their outputs, the interventions could therefore appear to change the same internal state. Their immediate effects on the residual stream are different. On held-out examples, the applied update is 2.19× larger than the matched register write and, when added to the same clean residual, produces a residual 1.68× larger. More importantly, the two updates have median cosine similarity 0.254, and only 5.7%–13.4% of the FV energy lies in the rank-16 register subspace. Thus, comparable behavioral success does not imply a comparable local edit. Counterfactual register editing writes within the identified causal subspace, whereas the successful FV intervention is a larger, mostly out-of-subspace update.

These measurements do not rule out another register, a distributed representation, or the possibility that later layers transform the FV update into the identified register. They demonstrate that successful FV steering is not itself a direct write into the register isolated previously. This motivates a broader question: when a steering vector changes behavior in standard tasks, does it recreate an activation that the model naturally uses, or control the answer through a different and unusually large perturbation? We test these alternatives next.

8 TESTING THE CORRESPONDENCE HYPOTHESIS

The register experiment gives a clear positive example: after we write a counterfactual value, the model applies a known transition to that value. We now ask what can be concluded when an additive steering vector changes an answer. Table 16 summarizes the evidence needed for three increasingly strong interpretations of an activation intervention.

8.1 COMPARING MODERN STEERING TECHNIQUES WITH THE MODEL'S OWN ACTIVATIONS

We continue our studies using Function Vector Steering. We use two models, and for each model, we choose a task, layer, and steering strength on separate calibration data before running the comparison below. The selected settings are **present-to-past conversion** in Qwen at block 7 with coefficient 15, and **English-to-Spanish translation** in Llama at block 8 with coefficient 10. Each evaluation uses 64 new prompts that do not appear in the construction or calibration data. Because these settings were chosen for a strong steering effect, the experiment tests how effective steering works rather than how often steering succeeds across tasks or layers.

For evaluation prompt i and block ℓ, let $h_{i,\ell}^{\text{shuf}}$ and $h_{i,\ell}^{\text{corr}}$ denote the hidden vector at the final prompt token after block ℓ for the shuffled and correct prompts, respectively. Let $\ell_\star$ be the selected intervention block, and let v be the calibrated Function Vector. All interventions below begin from the shuffled prompt; the corresponding correct prompt supplies hidden vectors for comparison.

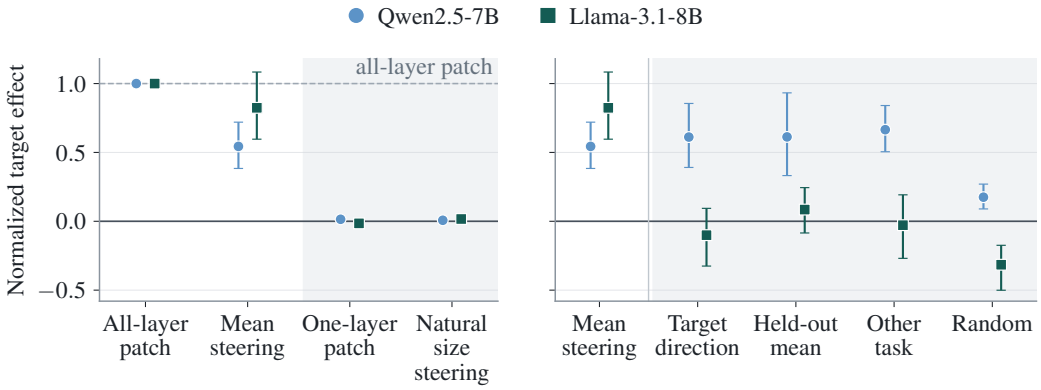
The register experiment established a strong form of local sufficiency: writing one counterfactual value at one token was enough for a known later operation to act on the edited value. The lexical tasks below do not provide an explicit state variable with a known update rule, so we cannot repeat that full semantic test. We instead test a weaker consequence of the same interpretation: whether successful steering can be explained as a one-token, one-block write of the hidden vector produced by the correct prompt. We compare five interventions:

1. **Repeated correct-prompt patch:** At the selected block and every subsequent block ℓ, replace the final-token hidden vector with $h_{i,\ell}^{\text{corr}}$, leaving earlier token positions unchanged. This provides a reference effect under sustained patching.
2. **One-block correct-prompt patch:** Replace $h_{i,\ell_\star}^{\text{shuf}}$ with $h_{i,\ell_\star}^{\text{corr}}$ once and let the remaining blocks proceed normally. This tests whether the correct-prompt value is locally sufficient to redirect computation.
3. **Mean steering:** Add the calibrated Function Vector v once at $\ell_\star$, measuring the behavioral effect of the standard steering intervention.
4. **Natural-scale mean steering:** Add v after rescaling it to $\|h_{i,\ell_\star}^{\text{corr}} - h_{i,\ell_\star}^{\text{shuf}}\|_2$. This tests whether steering remains effective at the scale of the natural correct-versus-shuffled activation change.
5. **Same-norm comparison directions:** Add the prompt-specific correct-minus-shuffled direction, a held-out average of that direction, another task's Function Vector, or a random direction, each rescaled to $\|v\|_2$. These controls test whether the effect is specific to the learned Function Vector.

The one-block patch tests whether the model's target activation is sufficient at the intervention site. Natural-scale steering tests whether the steering direction remains effective at the magnitude of the matched activation change. Together, these interventions test the two predictions of CH.

8.2 SCORING

For each prompt i, let $\Delta_i(c)$ be the change in the target-minus-alternative log-probability margin under intervention c, relative to the unedited shuffled prompt, and let G_i be the corresponding gain



(a) Target activation versus mean steering

(b) Specificity differs across models

Figure 3: **Effective steering does not reproduce the model's target activation at one layer.** (a) Patching target activations through all later layers is the normalized reference. Patching the target activation at only the tested layer and applying the steering vector at the scale of the natural activation change both have effects near zero. The conventional, much larger steering vector changes the answer in both models. (b) Llama responds to the task's steering direction but not to equally large comparison directions. Qwen responds to several structured directions, although not as strongly to a random direction. Points show ratios of paired mean margin changes; bars are paired 95% bootstrap intervals over 64 new prompts per model.

from the repeated correct-prompt patch. Across n evaluation prompts, the normalized effect is

$$E(c) = \frac{\frac{1}{n}\sum_{i=1}^{n} \Delta_i(c)}{\frac{1}{n}\sum_{i=1}^{n} G_i}.$$

Thus, $E(c) = 0$ denotes no average improvement and $E(c) = 1$ denotes the same average margin gain as the reference. Values above one are possible because the reference is not an upper bound. This score compares behavioral effects, not similarity between hidden vectors. We report paired 95% bootstrap intervals from 2,000 resamples; full definitions and normalization details are in Appendix E.3.

9 Steering does not reproduce the target activation

The register experiments show that a single local activation edit can be sufficient to redirect subsequent computation: once the state is changed, the model continues from the edited value. We now ask whether the activation targeted by Function Vector steering has the same kind of local sufficiency. In particular, does the model's own target-task activation work when transplanted once at that site, and does effective steering operate at the scale of this natural change?

9.1 The model's target activation is not sufficient

The all-layer target patch selects the target answer on all 64 prompts and increases the target-minus-alternative margin by 4.41 nats in Qwen and 2.64 nats in Llama, establishing the reference effect. Patching the matched target activation only at the tested layer produces almost none of this gain: its normalized effect is 0.014 [0.005, 0.023] in Qwen and −0.015 [−0.041, 0.005] in Llama (Figure 3a).

The natural-scale test also fails. The calibrated Function Vectors have norms 210.80 in Qwen and 20.22 in Llama, 77 and 26 times the median source-to-target changes of 2.74 and 0.78. Rescaling the vectors to those natural magnitudes reduces their normalized effects to 0.007 and 0.017. At the calibrated scale, they produce 0.544 and 0.824 of the all-layer patching gain.

Neither prediction of CH holds in the two tested settings: the matched target activation and natural-scale steering have little effect, whereas a much larger steering vector changes the answer. Successful steering therefore need not recreate the model's natural target activation at the tested layer.

9.2 THE TWO MODELS DIFFER IN STEERING SPECIFICITY

Because the calibrated steering vectors are much larger than the models' natural activation changes, we use equal-norm controls to test whether their effects depend on direction (Figure 3b). In Llama, the target task's Function Vector has a normalized effect of 0.824, while the four comparison directions range from −0.316 to 0.085; among the directions tested, the effect is therefore selective. Qwen behaves differently: three structured comparison directions have effects between 0.612 and 0.665, comparable to or larger than the target Function Vector's 0.544, although a random direction is weaker at 0.175. Thus, scale alone does not determine the effect, but specificity is model dependent. Even in Llama, specificity strengthens the mechanistic interpretation without showing that steering recreates a natural target activation or writes a state-like value.

9.3 ADDITIONAL TESTS NARROW THE INTERPRETATION

These tests do not identify what the large vector does, but they provide evidence against two additional simple explanations. Across two models and four lexical tasks, we add the vector at its usual layer and subtract it at progressively later layers. Early subtraction eliminates the task-specific effect; the effect survives only when subtraction occurs late, at depths where matched comparison directions also begin to affect the answer. We therefore find no evidence that the early steering pulse is quickly converted into a self-sustaining task state. When the underlying task is held fixed but the output labels are remapped, the unedited models follow the new mapping, whereas the steering effect does not reverse with it. Part of the effect remains tied to the same literal label, arguing against an abstract "correct answer" signal. Full results appear in Appendix F.

10 DISCUSSION

The controlled state-tracking experiments provide direct evidence for a causal register in the residual stream. Writing a counterfactual value $\tilde{s}$ into the low-rank component at the current-state token causes the model to apply the actual upcoming move m and predict $m(\tilde{s})$. The consequence of the same write changes with the upcoming move, while a conflicting continuation associated with the injected state does not determine the prediction. The counterfactual branch also persists during free-running generation, and the core result replicates in a second model family. Together, these tests establish a causal register: the value written at the trace token is read by later computation and updated according to the task rule. The causal effect saturates at rank one in Q_8 and rank two in D_8, although the update route is concentrated in Q_8 and more distributed in D_8. Thus, the dimensionality of a stored variable and the concentration of the computation that uses it are separate properties.

The Function Vector interventions present a different picture. Replacing the final-token residual with the exact residual from the matched prompt has almost no effect, and a Function Vector rescaled to the natural correct-versus-shuffled change is similarly ineffective. At its calibrated scale, the vector is effective, but its norm is 77 times the median natural change in Qwen and 26 times that change in Llama. Successful steering at these sites is therefore not well described as restoring the residual that the model naturally reaches on the target task. Successful interventions can still be *unnatural*, and the norm comparison quantifies one aspect of their departure from natural activation changes.

Subtracting the vector at an early downstream layer eliminates the task-specific effect; much of the effect survives only when subtraction is delayed until late in the network. Under this test, we therefore find no evidence that the initial perturbation is quickly converted into an independent, self-sustaining task state. Answer-remapping and equal-norm controls further show that the vector is not simply an abstract "correct answer" signal and that scale alone does not determine the effect.

11 CONCLUSION

Our experiments distinguish changing an answer from changing a state that later computation uses. In the controlled tasks, an activation edit supplies a value that subsequent operations use according to the task rule. Function Vector steering changes behavior but does not reproduce the model's natural target activation at the tested site. Establishing a state write therefore requires evidence about how later computation uses the edited value, beyond whether the intervention changes the answer.

AI USE STATEMENT

Generative AI tools assisted with experimental and analysis code, interpretation of results, manuscript editing, and preparation of figures and tables. The authors reviewed the AI-assisted work and take responsibility for the final text, methods, results, and claims.

REFERENCES

A. Conmy, A. N. Mavor-Parker, A. Lynch, S. Heimersheim, A. Garriga-Alonso. Towards Automated Circuit Discovery for Mechanistic Interpretability. NeurIPS, 2023. arXiv:2304.14997.

N. Elhage, N. Nanda, C. Olsson, et al. A Mathematical Framework for Transformer Circuits. Transformer Circuits Thread, 2021.

J. Engels, E. J. Michaud, I. Liao, W. Gurnee, M. Tegmark. Not All Language Model Features Are (One-Dimensionally) Linear. arXiv:2405.14860, 2024.

S. Feucht, T. Haklay, U. Bhalla, et al. Arithmetic in the Wild: Llama Uses Base-10 Addition to Reason About Cyclic Concepts. arXiv:2605.01148, 2026.

A. Geiger, H. Lu, T. Icard, C. Potts. Causal Abstractions of Neural Networks. NeurIPS, 2021.

A. Geiger, Z. Wu, C. Potts, T. Icard, N. D. Goodman. Finding Alignments Between Interpretable Causal Variables and Distributed Neural Representations. Causal Learning and Reasoning (CLeaR), 2024.

J. Hewitt, C. D. Manning. A Structural Probe for Finding Syntax in Word Representations. NAACL, 2019.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen. LoRA: Low-Rank Adaptation of Large Language Models. ICLR, 2022. arXiv:2106.09685.

G. Ilharco, M. T. Ribeiro, M. Wortsman, S. Gururangan, L. Schmidt, H. Hajishirzi, A. Farhadi. Editing Models with Task Arithmetic. ICLR, 2023. arXiv:2212.04089.

S. Kantamneni, M. Tegmark. Language Models Use Trigonometry to Do Addition. arXiv:2502.00873, 2025.

K. Li, A. K. Hopkins, D. Bau, F. Viégas, H. Pfister, M. Wattenberg. Emergent World Representations: Exploring a Sequence Model Trained on a Synthetic Task. ICLR, 2023a. arXiv:2210.13382.

K. Li, O. Patel, F. Viégas, H. Pfister, M. Wattenberg. Inference-Time Intervention: Eliciting Truthful Answers from a Language Model. NeurIPS, 2023b. arXiv:2306.03341.

B. Liu, J. T. Ash, S. Goel, A. Krishnamurthy, C. Zhang. Transformers Learn Shortcuts to Automata. ICLR, 2023.

S. Ma, Y. Luo, Zhangji, C. Xiao, A. Gao, W.-H. Huang, W. Wang, Q. Wu, X. Li, J. Wei, Q. Zhang. Hidden APIs in Language Models: Discovering Reusable Causal Interfaces from Forked Futures. arXiv:2607.27617, 2026.

K. Meng, D. Bau, A. Andonian, Y. Belinkov. Locating and Editing Factual Associations in GPT. NeurIPS, 2022. arXiv:2202.05262.

N. Nanda, L. Chan, T. Lieberum, J. Smith, J. Steinhardt. Progress Measures for Grokking via Mechanistic Interpretability. ICLR, 2023a. arXiv:2301.05217.

N. Nanda, A. Lee, M. Wattenberg. Emergent Linear Representations in World Models of Self-Supervised Sequence Models. BlackboxNLP, 2023b. arXiv:2309.00941.

M. Nye, A. J. Andreassen, G. Gur-Ari, et al. Show Your Work: Scratchpads for Intermediate Computation with Language Models. arXiv:2112.00114, 2021.

C. Olsson, N. Elhage, N. Nanda, et al. In-context Learning and Induction Heads. Transformer Circuits Thread, 2022. arXiv:2209.11895.

N. Prakash, T. Rott Shaham, T. Haklay, Y. Belinkov, D. Bau. Fine-Tuning Enhances Existing Mechanisms: A Case Study on Entity Tracking. ICLR, 2024. arXiv:2402.14811.

N. Prakash, N. Shapira, A. Sen Sharma, C. Riedl, Y. Belinkov, T. Rott Shaham, D. Bau, A. Geiger. Language Models Use Lookbacks to Track Beliefs. arXiv:2505.14685, 2025.

A. Syed, C. Rager, A. Conmy. Attribution Patching Outperforms Automated Circuit Discovery. arXiv:2310.10348, 2023.

E. Todd, M. L. Li, A. Sen Sharma, A. Mueller, B. C. Wallace, D. Bau. Function Vectors in Large Language Models. ICLR, 2024. arXiv:2310.15213.

A. M. Turner, L. Thiergart, G. Leech, D. Udell, J. J. Vazquez, U. Mini, M. MacDiarmid. Activation Addition: Steering Language Models Without Optimization. arXiv:2308.10248, 2023.

S. Venkatesh, A. Mahendran Kurapath. On the Non-Identifiability of Steering Vectors in Large Language Models. arXiv:2602.06801, 2026.

J. Wei, X. Wang, D. Schuurmans, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS, 2022. arXiv:2201.11903.

D. Wurgaft, C. Rager, M. Kowal, et al. Manifold Steering Reveals the Shared Geometry of Neural Network Representation and Behavior. arXiv:2605.05115, 2026.

J. Ye, L. Ran, Z. Yao, C. Wang, Y. Jiang, L. Hou, J. Li, L. Pan. Where Steering Signals Come From: Activation Source Selection in Activation Steering. arXiv:2607.25270, 2026.

Y. Zhang, et al. Finite State Automata Inside Transformers with Chain-of-Thought: A Mechanistic Study on State Tracking. ACL, 2025. arXiv:2502.20129.

A. Zou, L. Phan, S. Chen, et al. Representation Engineering: A Top-Down Approach to AI Transparency. arXiv:2310.01405, 2023.

A TASK AND EXPERIMENTAL SETUP

The register experiments use a known transition rule to distinguish changing a state from changing an answer. This appendix collects the state-space construction, prompt and data specifications, and model-training settings.

A.1 STATE SPACE AND TOKEN VOCABULARY

The two move systems are the quaternion group Q_8 and the dihedral group D_8, the two non-commutative groups of order eight. We use them because each is a central extension of the Klein four-group $V = \mathbb{Z}/2 \times \mathbb{Z}/2$ by $\mathbb{Z}/2$:

$$1 \longrightarrow \mathbb{Z}/2 \longrightarrow G \longrightarrow V \longrightarrow 1, \qquad G \in \{Q_8, D_8\}.$$

This gives every state $g \in G$ a two-part description. The image of g in the quotient V is the *visible coordinate*: a four-way value that a model can read directly from the move sequence, because the quotient multiplication is commutative. The remaining $\mathbb{Z}/2$ factor is the *hidden coordinate*: a single bit fixed by the group's two-cocycle, which records the order in which moves were applied. Q_8 and D_8 are distinguished precisely by their cocycle, so a hidden-coordinate result that holds for both is a property of the supervision and not of one cocycle.

Order sensitivity. The hidden bit is what makes the task order-sensitive. For two moves a, b whose visible parts commute, the products ab and ba share a visible coordinate but differ in the hidden coordinate whenever the cocycle is nontrivial on that pair. A model that tracks only the visible coordinate, or only the move multiset, cannot distinguish ab from ba; a model that carries the hidden coordinate can. The held-out loop tests are sequences that return to the identity in the quotient but differ in the hidden bit, so they pass only if the order-sensitive coordinate was learned.

Labels. In figures and tables a state is written as $s = (v, h)$: a visible value $v \in \{00, 01, 10, 11\}$ and a hidden bit $h \in \{0, 1\}$, written `v|h` (for example `11|1`). The model itself reads and writes a small fixed vocabulary of single tokens, a *region* letter for the visible value, a *phase* letter for the hidden bit, and a move letter for each generator, chosen so that every token is a single subword and the current-state and next-state positions are fixed across prompts (Appendix A.2). Table 1 records the exact correspondence we use to relabel that raw vocabulary as coordinates throughout the paper; these letters are the `J`, `M`, `F`, `C` (and `P`, `R`) that appear in the model's own input and output. The labels are a presentation device for the eight states; the operational content is the quotient and the central bit defined above.

Table 1: **Token vocabulary and its coordinate relabeling.** The four region letters and two phase letters are the eight raw state tokens (`J`, `M`, `F`, `C`, ...) the model reads and writes; the move letters `Q`, `T`, `W`, `Z` are the four generators a, b, a^{-1}, b^{-1}. The inverse moves `W`, `Z` project to the same visible bit flips as `Q`, `T` but differ in their effect on the hidden bit, so the quotient walk is identical across Q_8 and D_8 while the hidden bit is not.

Region v (visible)		**Phase** h (hidden)		**Move** (generator)	
token	coord.	token	bit	token	action
`C`	`00`	`P`	0	`Q`	flip 1st bit (a)
`F`	`10`	`R`	1	`T`	flip 2nd bit (b)
`J`	`01`			`W`	a^{-1}
`M`	`11`			`Z`	b^{-1}

Figure 4 shows the cocycle difference on a single worked example: the same move sequence reaches the same visible cells in Q_8 and D_8, but the hidden bit diverges in the middle of the walk.

Figure 5 makes the same difference structural: with the hidden bit drawn as a vertical axis, the move T stays within an h level in D_8 (order 2) but threads both levels in Q_8 (order 4); the quotient walk on the visible cells is identical in both groups, and only the hidden-bit update differs.

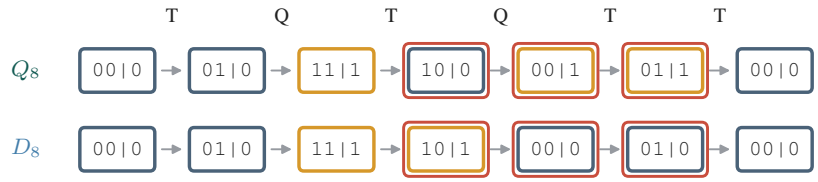


Figure 4: **Same moves, same visible path, different hidden bit.** The move sequence `T Q T Q T T` from `00 | 0` produces the same sequence of visible coordinates in Q_8 and D_8 (the quotient walk is identical), but the hidden bit h (amber $h{=}1$, steel $h{=}0$) diverges where boxed. The cocycle that separates the two groups, on one worked example.

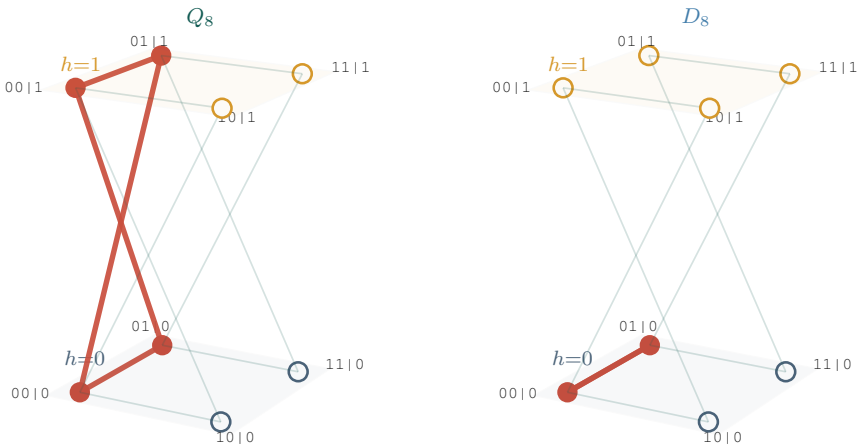


Figure 5: **How the Q_8 and D_8 cocycles differ.** The eight states are the corners of a cube whose vertical axis is the hidden bit h; the four visible cells sit in each level. The move Q (teal) is identical in both groups. The move T (coral orbit from `00 | 0`) stays within one level in D_8, so it has order 2; in Q_8 it flips h and threads both levels, so it has order 4. The quotient walk on the visible cells is the same in both groups; only the hidden-bit update differs, and that extra turn through the hidden bit is the cocycle that separates them.

A.2 Prompts, datasets, and evaluation splits

Prompt format. Each item presents an initial state and a sequence of moves, and asks for the resulting state. In the running-state condition the target sequence emits the intermediate state after each move before the final answer; in the final-answer condition the target is the final answer only. Moves and states are written with a small fixed vocabulary so that the current-state token and the next-state readout occur at known positions, which is what makes the register edit and the edge interventions positionally well defined; the worked example in Figure 6 shows the format.

Sequences and splits. Items are generated by sampling move sequences over the group and computing the induced state trajectory. The supervised fine-tuning corpus for each group (Q_8, D_8) holds 11,346 training items, 3,782 distinct move sequences of length 1–6, phase-balanced within each length and move-multiset bucket, each rendered under three instruction paraphrases, together with 856 held-out validation items (a single held-out template on disjoint sequences). We hold out a disjoint split of items for all reported measurements, and the localization analysis uses a second, independently sampled split to test replication. Counterfactual items for the register edit are constructed by pairing each held-out current state with its same-visible, opposite-hidden partner; conflicting-continuation items additionally require a real occurrence of the partner state whose continuation disagrees with the move-applied target, so the two outcomes are discriminating by construction.

Discriminating items. The selectivity metrics are evaluated only on items where the two targets being compared differ, so that a difference in outcome is attributable to the intervention. Each reported selectivity metric uses $n = 400$ held-out discriminating items; the layer-resolved decodability of Figure 7 uses $n = 600$ held-out items per layer. Move sequences are sampled to balance the eight states and the four generators; the full dataset, with per-state and per-move counts, is released with the code.

Prompt templates. The model reads the raw single-token vocabulary of Table 1. Held-out items share the prefix `Walk. Start region {region} phase {phase}. Moves {moves}` and end with `Final region then phase:` (final-answer arm) or `State trace, then final region and phase:` (running-state arm, whose target emits each intermediate `region phase` pair before the final answer). Training uses three paraphrases of this instruction; the held-out split uses the template above, so the evaluation wording is disjoint from training.

A.3 Use of Large Language Models

We used large language models for routine coding assistance and for proofreading and formatting the manuscript.

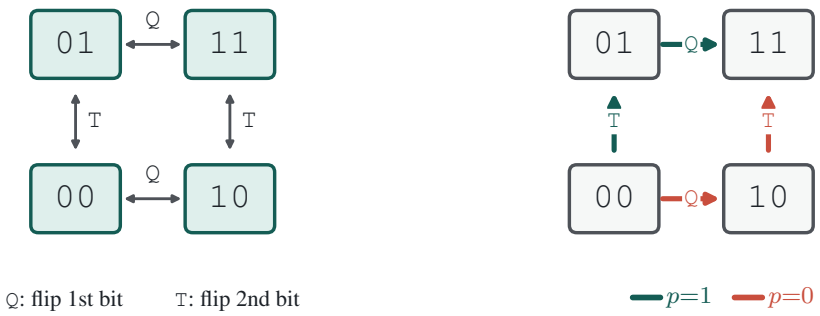
A.4 Models, training, and intervention sites

Base model. The primary three arms are built from `Qwen2.5-Coder-7B`; the second-family replication (Appendix B.2) trains the same arms from `Mistral-7B-v0.3`. The base arm is the unmodified pretrained model.

Fine-tuning. The final-answer and running-state arms are produced by low-rank adaptation of the base model on the task. The two arms use the same move sequences and the same final answers and differ only in whether the intermediate state is present in the training target. This matched design is what lets us attribute the causal use of the register to trace supervision rather than to task exposure or to optimization.

Compute Resources All experiments were run on RunPod cloud instances equipped with NVIDIA A100 or H100 GPUs, each with 80 GB of GPU memory. Each experimental run used 1 GPU and required approximately 1 hour.

Intervention sites. The register edit acts on the residual stream at the current-state token at a mid-network layer (layer 12), selected and locked on a validation split before the causal test; results are reported at this layer and replicated at a neighboring layer (Appendix C.5). The localization



(a) Each move flips one visible bit

(b) Order sets the phase bit

Prompt (user).

```
Walk.  Start state 00|0.  Moves T Q T Q T T
State trace, then final state:
```

Running-state output (model).

```
01| 0 → 11|1 → 10| 0 → 00| 1 → 01| 1 → 00| 0
Final:  00|0
```

Figure 6: **The task as a walk on a small state space, with the hidden coordinate and an edit site highlighted.** A state is a visible coordinate $v \in \{00, 01, 10, 11\}$ (one of four cells of the 2×2 grid) and a hidden bit h, written $v \mid h$. (a) Each move flips one visible bit, so v is a function of the move multiset and easy to read. (b) The hidden bit is order-dependent: the same moves in a different order can reach the same cell with the opposite h, so it must be tracked as the sequence is consumed. An example prompt and the model's running-state output are shown in these coordinates (the model's literal single-token vocabulary and its mapping to $v \mid h$ are in Table 1). To run the counterfactual edit of Section 4 we take one current-state token (amber, here `11|1`), replace a rank-16 subspace of its residual stream so the hidden bit reads the opposite value (`11|1` → `11|0`) while the printed token is unchanged, and read the model's next state: if the register is executable, the next hidden bit follows the move applied to the edited state, not the original one.

analysis intervenes on attention edges into the next-state readout and on low-rank adapter bands across a fixed range of layers. Table 2 collects the settings needed to reproduce the reported measurements.

Table 2: **Experimental specification** for the reported measurements. Per-state and per-move balance, exact prompt templates, and dataset item counts are released with the code and data.

Component	Setting
Base model	`Qwen2.5-Coder-7B` (28 layers, $d_{\text{model}} = 3584$)
Fine-tuning (LoRA)	rank 16, $\alpha = 32$; target modules `q,k,v,o,gate,up,down_proj` (all layers)
Optimization	learning rate 10^{-4}, 4 epochs, warmup ratio 0.05, max sequence length 256
Register edit	rank-16 subspace at the current-state token, `resid_pre`, mid-network layer 12 (validation-selected edit site; replicated at a neighboring layer)
Localization scan	38 candidate components (attention edges into the next-state readout and low-rank adapter bands) over layers 12–27; the Q_8 winner is the layer-22 edge and the D_8 frozen set is layers $\{19, 22, 23, 25\}$
Evaluation	$n = 400$ held-out discriminating items per metric per split; Q_8 localization on two independent splits, D_8 route on ten reported splits (eight held-out confirmatory); layer-resolved decodability at $n = 600$ per layer

Reproducibility. The anonymous review supplement contains the source code, source CSV/JSON tables, and figure-generation scripts needed to reproduce every number in the paper.

B Causal register results and replication

We first collect the evidence that the edited value is used under the supplied transition rule, then test whether that causal chain transfers to a second model family and to generated traces. The second-family and free-running tests, together with the route-null and subset analyses in Appendix C.4, were conducted as follow-up analyses after the main results.

Register measurements are reported on held-out items. Selectivity is Counterfactual Update Selectivity (CUS, Section 4); item-level intervals are 95 percent bootstrap intervals over held-out items, while the D_8 four-edge route interval is computed across the ten reported splits (eight held-out confirmatory).

B.1 Supervision controls and the core causal chain

The main analysis focuses on the model trained with full running-state supervision. To separate the effect of exposing intermediate states from task exposure alone, we also evaluate two matched controls. The *pretrained* arm is the unmodified base model. The *final-answer-only* arm is fine-tuned on the same move sequences and final states as the full-supervision arm, but its target contains only the final state. The *full running-state* arm instead emits the state after every move before giving the final answer. Thus, the two fine-tuned arms see the same inputs and final answers and differ only in whether intermediate states appear in the training target. Exact model and optimization settings are given in Appendix A.4.

Readout comparison. At the edited current-state token, the hidden coordinate is the literal trace token and therefore decodes at 1.00 in every arm; edit-site readability does not distinguish the models. The relevant comparison is the *next*-state coordinate, which must be computed rather than read from the token. In Q_8, its maximum layerwise probe accuracy is 0.5350 in the pretrained model and 0.5150 in the final-answer-only model, but rises to 0.9650 in the full running-state model. In D_8, the pretrained and final-answer-only models already contain a partial next-state readout, reaching 0.7483 and 0.7567, while the full running-state model reaches 0.9633. The D_8 result should therefore be interpreted as converting an existing partial readout into an editable, executable register rather than creating the readout from nothing (Prakash et al., 2024).

Additional downstream readout. At the counterfactual-edit site (layer 12, `resid_pre`, the current hidden-bit token), the hidden coordinate is the literal trace token and decodes at 1.000 in every arm (base, final-answer, and running-state, in both Q_8 and D_8). Edit-site readability therefore does not distinguish the arms. The arm-distinguishing quantity is the *next*-state coordinate the model

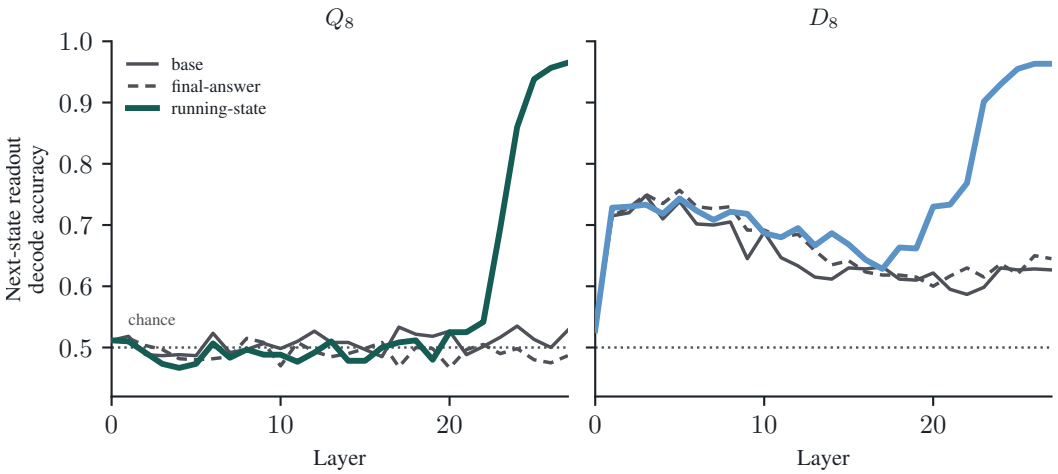


Figure 7: **Decodability of the next-state hidden coordinate by layer.** Linear-probe accuracy at each layer in the pretrained, final-answer-only, and full running-state models (chance 0.5). In Q_8, only the full running-state model decodes the coordinate, and only in the last layers. In D_8, the two auxiliary arms already decode it partially in early layers, but the full running-state model reaches a higher value in late layers. $n = 600$ held-out items per point.

must compute; a late-layer probe of it (`phase_read_resid_post`, layer 27) reaches 0.972136 [0.958204, 0.984520] in Q_8 and 0.976266 [0.963608, 0.987342] in D_8 for the running-state model, reported as a downstream readout rather than an edit-site measurement.

Causal comparison. Readability alone does not establish that the model uses the coordinate. Under the rank-16 counterfactual register edit, the pretrained and final-answer-only Qwen models remain at chance on the causal Use test, whereas the full running-state model reaches register agreement 0.80 in Q_8 and 0.91 in D_8. The same three-arm comparison is repeated for `Mistral-7B-v0.3`; its exact per-arm Use and selectivity results are reported in Appendix B.2.

Move-specific and conflicting-continuation controls. Move-swap selectivity is +0.5745 in Q_8 and +0.6768 in D_8. The conflicting-continuation control in Q_8 gives 0.7975 for following the current move against 0.2025 for following the injected continuation, a selectivity of +0.5950 with interval [0.5150, 0.6750]; a single injected occurrence is as effective as the constructed counterfactual. The same control in D_8 gives 0.9025 against 0.0975, a selectivity of +0.8050 with interval [0.7450, 0.8600] ($n = 400$ held-out items in each antileak row), so the edit supplies a state and not an answer in both groups.

Supporting layer-stitching and transfer checks. In a layer-stitching check, the early and middle layers of the running-state model drive the later layers of a weaker model, while a weak early computation is not restored by strong later layers. In a base-coordinate transfer check, the best base-to-running decoder reaches 0.537 full-state accuracy against a running self-decoding ceiling of 0.557 and a chance rate of 0.125, with retention 0.953; a surface-feature control reaches 0.277, above the 0.25 threshold, so this check is supportive but mixed and does not by itself exclude a readout shortcut. These results corroborate that the supervision builds, rather than only routes, the computation, and they are not part of the headline causal chain.

B.2 The core chain in a second model family

We repeated the core counterfactual-register chain on `Mistral-7B-v0.3`, trained with the same running-state training procedure and scored on held-out items with a rank-16 register edit, read out at the layer chosen by the predeclared earliest-readable-current-phase rule. We did not attempt circuit localization in this model. Because the current-state token is teacher-forced it is linearly readable in every arm (Read = 1.00 throughout), so Read does not distinguish the arms here; *use* does. Only the

Table 3: **The counterfactual register chain, with a reference level for every row.** Each row reports a different quantity, named in the row, so the table is read row by row and not down a single scale. The *null* column gives the value that same quantity takes when the effect is absent, a chance level or a matched control, and each Q_8/D_8 entry is read against it. The edit-site check verifies the hidden bit is readable at the edited token, which is the literal trace token and so decodes at 1.00 in every arm; it is a sanity check, not evidence for trace supervision, and the causal chain begins with Use. Use is the probability that the next hidden coordinate matches the move applied to the *edited* state (random and orthogonal-complement edits give 0.02); Move-specific and Conflicting continuation are selectivity gaps that are 0 when the model ignores the edit; Localize is the fraction of the counterfactual update removed when its attention route is ablated. The conflicting-continuation control passes in both groups; the D_8 update route is localized across ten reported splits, eight of them held-out confirmatory (mean 68.7% reduction, 95% CI $[66.9, 70.4]$). The next-state coordinate that the model must *compute*, rather than read off the token, is the arm-distinguishing readout in Figure 7. Full tables and intervals are in Appendices B and C.

Step (held-out), with the quantity it reports	**null**	Q_8	D_8
Edit-site check: is the literal hidden-bit token readable? (accuracy)	1.00	1.00	1.00
Use: does the edit set the next state? $\mathbb{P}(\hat{h}_{t+1} = h(m \cdot \tilde{s}))$	0.02	0.80	0.91
Move-specific: does it follow the supplied move? (gap)	0	+0.57	+0.68
Conflicting continuation: current move minus injected continuation	0	+0.59	+0.81
Localize: fraction of CUS removed by the update route	0	94%	68.7%

running-state arm passes Use and the two selectivity controls; the base and final-answer arms decode the state but stay near chance on Use and near zero on the controls (Table 4). The executable register chain therefore replicates in a second model family.

Table 4: **Core counterfactual-register chain in `Mistral-7B-v0.3`.** Exact per-arm means with 95 percent item-bootstrap intervals. Use and Conflicting continuation use $n = 2000$; the move-specific column uses the discriminating subset ($n = 1334$ for Q_8, $n = 967$ for D_8). Circuit localization was not evaluated in this model.

Group	Arm	Use	Move-specific	Conflicting continuation
Q_8	running-state	0.9265 [0.9150, 0.9375]	0.8471 [0.8186, 0.8756]	0.8530 [0.8300, 0.8750]
Q_8	base	0.5010 [0.4790, 0.5225]	−0.0060 [−0.0600, 0.0465]	0.0020 [−0.0420, 0.0450]
Q_8	final-answer	0.5050 [0.4830, 0.5270]	0.0105 [−0.0435, 0.0630]	0.0100 [−0.0340, 0.0540]
D_8	running-state	0.9430 [0.9325, 0.9525]	0.8428 [0.8077, 0.8759]	0.8860 [0.8650, 0.9050]
D_8	base	0.5355 [0.5130, 0.5575]	−0.0527 [−0.1148, 0.0093]	0.0710 [0.0260, 0.1150]
D_8	final-answer	0.4425 [0.4210, 0.4645]	0.0052 [−0.0569, 0.0693]	−0.1150 [−0.1580, −0.0710]

B.3 Free-running register edits and branch persistence

This is the free-running test summarized in Section 5.1. On the `Qwen2.5-Coder-7B` running-state models we let the model greedily generate the running trace, retain only rollouts whose generated assistant prefix matches the correct trace token-for-token through the current hidden-bit token (the retained fraction is the *coverage*), and apply the same rank-16 layer-12 `resid_pre` same-visible/opposite-hidden edit to the model's own generated current-state token. We score the one-step counterfactual update, the move-specific and conflicting-continuation selectivities, and multi-step *persistence*: after the edit we continue the greedy rollout and ask whether each later hidden state, and the final answer, follows the edited branch rather than the original branch. The protocol, exact-prefix filter, and a success threshold of 0.70 for Use were fixed before scoring; D_8 passes the threshold and Q_8 does not (one-step Use 0.66), and we report both. Intervals are 95% item-level bootstraps.

Table 5: **Free-running counterfactual register editing on `Qwen2.5-Coder-7B`.** The running-state model generates its own trace; only rollouts whose generated prefix is exactly correct through the edited token are scored. The hidden-bit edit redirects the next state and the redirection persists over the next three states and into the final answer, cleanly in D_8 and partially in Q_8 (one-step Use $0.66 < 0.70$).

Metric	Q_8	D_8
Coverage (exact-prefix filter)	0.464	0.768
One-step Use, $\mathbb{P}(\hat{h}_{t+1} = h(m \cdot \tilde{s}))$	0.660 [0.597, 0.723]	0.902 [0.871, 0.934]
Move-specific selectivity	+0.260 [0.099, 0.428]	+0.637 [0.521, 0.743]
Conflicting continuation selectivity	+0.376 [0.183, 0.570]	+0.385 [0.180, 0.590]
Branch persistence, k=2	+0.466 [0.343, 0.590]	+0.722 [0.639, 0.799]
Branch persistence, k=3	+0.403 [0.269, 0.537]	+0.724 [0.627, 0.813]
Branch persistence, k=4	+0.402 [0.231, 0.556]	+0.702 [0.590, 0.814]
Final-answer edited-branch agreement	0.723 [0.660, 0.782]	0.845 [0.808, 0.882]

C REGISTER DIMENSIONALITY AND UPDATE LOCALIZATION

A compact register need not be updated by a single concentrated circuit. We bring together the localization protocol, the complete Q8 and D8 results, and the controls that test the identified routes.

C.1 CANDIDATE SCAN, DECISION RULES, AND ROUTE SELECTION

The register edit and its control conditions (full residual, random subspace, orthogonal complement, and the null no-patch and identity conditions), and the metrics, register agreement and Counterfactual Update Selectivity (CUS) with the move-swap and conflicting-continuation selectivities, are defined in Section 4. An ablation masks a candidate component during the forward pass and rescores CUS; the matched control routes the same component from the visible coordinate rather than the hidden coordinate; and a restoration recovers the component's contribution from the clean edited run. The following criteria determine when a component, or a set of components, mediates the update.

For a *single localized mediator* we require a reduction of at least 80 percent with the matched control preserved (reduction at most 20 percent), a restoration that recovers at least 70 percent of the lost selectivity, and ordinary unedited behavior intact. When no single component meets this bar, the update may still be mediated by a distributed pathway, a small fixed set of components acting together. For a distributed pathway the appropriate threshold is lower, because the effect is shared: a joint ablation of a fixed edge set counts as *strong* at a reduction of at least 70 percent with matched controls preserved and restoration successful, and *partial* between 40 and 70 percent. Applying the 80 percent single-edge bar to a genuinely distributed pathway would misreport it as a failure.

These percentages are *predeclared conventions*: each is the fraction of the baseline selectivity that an ablation removes, and the cutoffs were fixed before scoring rather than derived from any theory. They serve as a decision rule, in the role a significance level plays for a statistical test, so the choice cannot be tuned after seeing the data. The conclusions do not turn on their exact values: the Q_8 edge removes 94 percent of the selectivity, well above the 80 percent bar, and the strongest single D_8 edge removes 24.2 percent, well below it, so both verdicts survive any reasonable choice of cutoff. The single boundary case is the D_8 route, whose joint reduction averages 68.7 percent across ten reported splits (95 percent CI 66.9–70.4), below the 70 percent strong/partial line, so we report the route conservatively as *partial* rather than strong.

The distributed pathway for D_8 was tested with a nested ablation of a fixed four-edge set, frozen before scoring; it reaches the partial tier (Appendix C.3).

D_8 route selection protocol. The four-edge set (the L22, L25, and L23 current-hidden-bit edges and the L19 move edge) was selected on the single-ablation candidate scan together with two selection splits (seeds 0–1), then frozen before the confirmatory evaluation. The confirmatory evaluation added eight independently sampled held-out splits (seeds 2–9); we fixed $N = 10$ in advance and report all ten splits, summarized by the means in Figure 9 and released in full with the code. No edge was

added, removed, reordered, or rescored after observing the ten-split outcomes. Because two of the ten splits informed selection, the eight held-out splits alone average 68.4 percent, indistinguishable from the ten-split mean, so the estimate is not inflated by selection.

We localize the update with the same single-ablation scan in both groups: a fixed set of 38 candidate components on the path to the next-state readout (attention edges and low-rank adapter bands) each ablated in turn under the register edit. The two groups separate across the whole search space (Figure 8): in Q_8 one component, the layer-22 current-state hidden-bit edge, removes 94 percent of the selectivity while every other candidate removes at most 15 percent; in D_8 the same edge is the strongest candidate but removes only 24 percent, and the rest are far lower, with no single mediator.

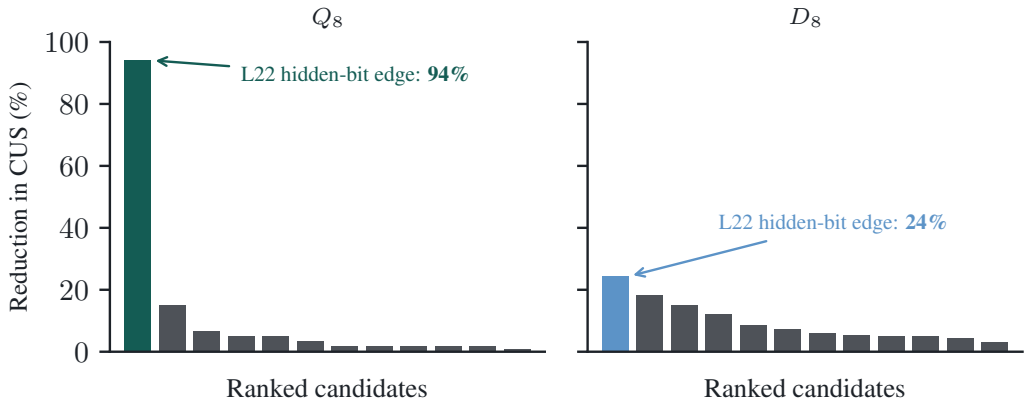


Figure 8: **The update is concentrated in Q_8 and distributed in D_8.** Bars show the leading candidates from the fixed 38-candidate scan, ranked by the reduction in Counterfactual Update Selectivity (CUS) when each is ablated alone; only the leading candidates are plotted, and the remaining lie in the near-zero tail (the full 38-candidate ranking is released with the code). In Q_8 one component dominates (the layer-22 current-state hidden-bit edge, teal; 94 percent), and the next-largest removes only 15 percent. In D_8 the same edge is the strongest candidate (blue) but removes only 24 percent, and the remaining candidates decline gradually, with no isolated mediator.

C.2 A SINGLE ATTENTION EDGE MEDIATES THE Q_8 UPDATE

In Q_8, a single attention edge mediates the counterfactual update. Scanning a fixed candidate set on the path to the next-state readout, a single attention edge stands out, in the attention-head circuit terms of the transformer-circuits analyses (Elhage et al., 2021; Olsson et al., 2022): the edge from the current-state hidden coordinate to the next-state readout at layer 22. Its ablation removes about 94 percent of the baseline selectivity, driving CUS from 0.595 to 0.035; its restoration recovers CUS to 0.595; and the matched control edge, routed from the visible coordinate to the same readout, leaves CUS near 0.59. Ordinary unedited behavior is preserved. This edge is the isolated winner of the scan: the next-best candidate removes about a sixth as much (Figure 8). The localization reproduces on an independent data split, where the same edge is again the isolated winner and again meets the full criterion, so it is not an artifact of one split. Figure 9 shows the ablation, control, and restoration for the Q_8 edge and for the D_8 route.

C.3 THE FROZEN D_8 ROUTE AND ITS REPLICATION ACROSS SPLITS

In D_8 the register replicates: editing it redirects the computation at 0.91 register agreement, with a baseline CUS of 0.825, and the move-swap and conflicting-continuation controls hold. The update route, however, is distributed rather than concentrated in a single edge. No individual candidate passes the single-edge criterion, and the layer-22 edge that mediates the Q_8 update removes only about a quarter of the D_8 selectivity (Figure 11b). We therefore test a fixed set of four edges, selected on the candidate scan and two selection splits, then frozen, in which the current-state hidden bit reaches the next-state readout through layers 22, 25, and 23 and the move token enters at layer 19. Ablating the set removes a mean of 68.7 percent of the selectivity across ten reported splits, eight of them

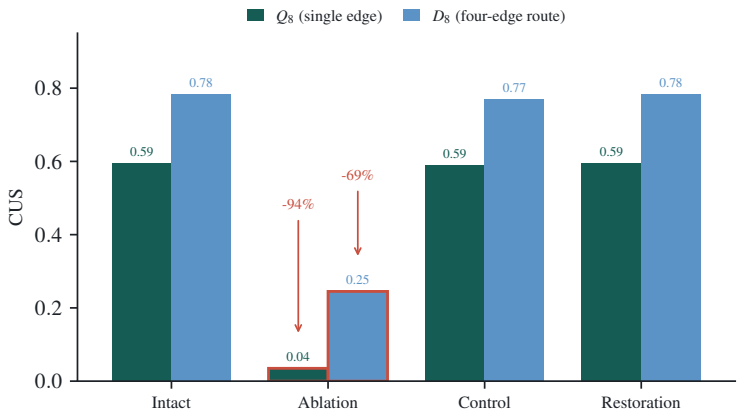


Figure 9: **Ablation and restoration validate the update routes.** Counterfactual Update Selectivity (CUS) for Q_8 (teal; the single layer-22 edge) and D_8 (blue; the frozen four-edge route, mean over ten reported splits, eight held-out confirmatory) under the intact edit, ablation of the route, a matched control, and restoration. Ablating the route drives CUS to near zero in both groups (a 94 percent reduction in Q_8 and 68.7 percent in D_8, highlighted), while the matched control and the restoration hold near the intact baseline.

Table 6: **Localization in Q_8 replicates across splits.** Ablating the layer-22 edge that carries the hidden coordinate into the next-state readout removes more than 90 percent of the selectivity on both splits; the matched control edge is preserved; the restoration recovers the selectivity. Ordinary unedited behavior, $\mathbb{P}(\hat{h}_{t+1} = h(m \cdot s))$, is preserved at 0.82 and 0.79. The edge is the isolated winner of the candidate scan; the next-best candidate removes 0.090 of selectivity on split 1 and 0.110 on split 2.

Q_8 **localization (layer-22 edge)**	baseline	ablation	matched control	restoration
CUS, split 1	0.595	0.035	0.59	0.595
CUS, split 2 (replication)	0.525	−0.050	0.53	0.525
relative reduction	—	0.94 / 1.10	≤ 0.02	restored ≥ 0.99

independent held-out confirmatory splits (95 percent CI 66.9 to 70.4): a replicated partial localization, consistent across splits but below the strong threshold met in Q_8. Across all ten splits both matched control sets preserve the effect, the clean restoration returns it to baseline within reported precision, and ordinary behavior stays intact (Tables 7 and 8).

The four-edge route is also unusually strong among same-size alternatives. Among 1000 random four-edge sets drawn from the 32 literal attention edges in the same candidate landscape, the median CUS removed is 0.055 and the 95th percentile is 0.254; the frozen route, which removes 0.532 on the held-out splits, lies at the 99.6th percentile (Appendix C.4). A subset analysis shows the route is distributed rather than single-edge: the strongest individual edge (layer 22) removes only 0.212, the full route removes 0.532, and the full route exceeds the sum of its singleton effects. This is a joint route with a dominant late current-state edge, not four equal mediators (Appendix C.4).

Table 7: **The D_8 update is partially localized to a distributed pathway.** A nested ablation of the fixed four-edge set E_4 (selected on the candidate scan and two selection splits, then frozen) removes a mean of 68.7 percent of the selectivity across ten reported splits (eight held-out confirmatory; 95 percent CI 66.9–70.4; the accumulation shown is the seed-0 selection split): a replicated partial localization that does not meet the strong single-edge criterion. The current-state hidden bit routes through layers 22, 25, and 23 into the next-state readout, and the move token enters at layer 19. Across all ten splits both matched controls preserve the selectivity (the quotient-source and off-target destination controls stay near 0.82), the clean restoration recovers it, and ordinary unedited behavior is intact ($\mathbb{P}(\hat{h}_{t+1} = h(m \cdot s)) = 0.88$). No single edge passes the single-edge criterion (the largest, L22, removes only 24 percent), so the D_8 update is distributed where the Q_8 update is concentrated.

D_8 **cumulative ablation** (baseline CUS $= 0.825$)	ablation CUS [95% CI]	reduction
E_1: L22 current hidden bit	0.625 [0.550, 0.700]	24.2%
E_2: + L19 move	0.575 [0.495, 0.655]	30.3%
E_3: + L25 current hidden bit	0.555 [0.470, 0.635]	32.7%
$\boldsymbol{E_4}$: + L23 current hidden bit	**0.265** [**0.170**, **0.360**]	**67.9%**

Table 8: **Ten-split D_8 frozen-route results.** Seeds 0–1 are the two selection splits; seeds 2–9 are held-out confirmatory. The frozen four-edge route has mean reduction 68.69% with split-bootstrap 95% CI [66.88%, 70.35%]. The clean restoration equals the reported baseline CUS in all ten splits; the quotient-source and off-target controls preserve selectivity in every row.

Seed	Split class	Base CUS	Ablated CUS	Quotient ctrl	Off-target ctrl	Restoration	Behavior	Reduction
0	selection	0.825	0.265	0.835	0.820	0.825	0.8750	67.88%
1	selection	0.790	0.225	0.790	0.755	0.790	0.8875	71.52%
2	held-out confirmatory	0.780	0.230	0.790	0.775	0.780	0.9025	70.51%
3	held-out confirmatory	0.725	0.230	0.725	0.695	0.725	0.8800	68.28%
4	held-out confirmatory	0.775	0.245	0.785	0.765	0.775	0.8825	68.39%
5	held-out confirmatory	0.795	0.235	0.795	0.780	0.795	0.8800	70.44%
6	held-out confirmatory	0.795	0.270	0.805	0.775	0.795	0.8750	66.04%
7	held-out confirmatory	0.785	0.210	0.790	0.770	0.785	0.9050	73.25%
8	held-out confirmatory	0.780	0.290	0.790	0.790	0.780	0.8800	62.82%
9	held-out confirmatory	0.775	0.250	0.785	0.760	0.775	0.8725	67.74%

C.4 Matched random routes and the contributions of individual edges

Random four-edge comparisons. On the eight held-out confirmatory splits the frozen four-edge route E_4 removes a mean 0.532 CUS. We compare it to 1000 random four-edge sets drawn from the 32 literal attention edges in the same candidate landscape; the broad LoRA-band candidates are excluded because they are not edge-comparable. The null has median 0.055 and 95th percentile 0.254, and E_4 lies at the 99.6th percentile (Figure 10). The route is thus unusually strong among same-size attention-edge sets: it is not the case that any four plausible edges remove two-thirds of the effect. We do not claim E_4 is the single most destructive four-edge set (the null maximum, 1.022, exceeds it) only that it is an unusually strong route that also passes the matched-control and restoration checks of Appendix C.3. Conditioning the null further on E_4's edge signature (three current-hidden-bit edges and one move edge into the next-state readout) leaves E_4 at the 99.0th

percentile among same-signature random four-edge sets (median 0.093, 95th percentile 0.336), so the route is unusually strong even against type- and destination-matched alternatives.

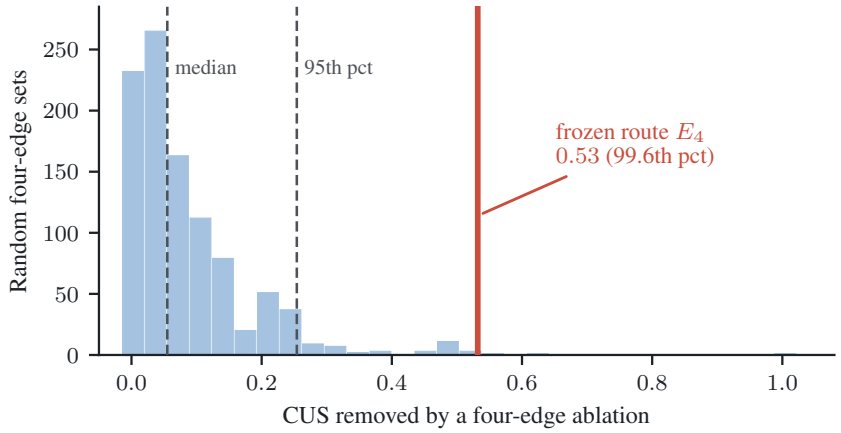


Figure 10: **The D_8 four-edge route is unusually strong among same-size edge sets.** Distribution of CUS removed by 1000 random four-edge sets drawn from the 32 literal attention edges in the candidate landscape (broad LoRA-band candidates excluded). The frozen route E_4 (coral) removes 0.532 and lies at the 99.6th percentile; the null median is 0.055 and the 95th percentile is 0.254. A single random set removes more than E_4, so the claim is percentile and route validation, not uniqueness.

Subset and leave-one-out analysis. Table 9 reports, for each edge in E_4, its effect ablated alone and its leave-one-out contribution (the drop in the full-route effect when that edge is removed). The strongest individual edge, the layer-22 current-hidden-bit edge, removes only 0.212 CUS, while the full route removes 0.532; no singleton approaches the route. The full route also exceeds the sum of the singleton effects (0.472), and the layer-22 and layer-23 edges carry the largest conditional contributions. The D_8 update is therefore distributed and partly synergistic: a joint route with a dominant late current-state edge, not single-edge mediation and not four equal independent effects.

Table 9: **Subset and leave-one-out anatomy of the D_8 four-edge route**, on the eight held-out confirmatory splits. "Alone" ablates only that edge; the leave-one-out drop is the reduction in the full-route effect when that edge is removed from E_4. Entries are mean CUS removed.

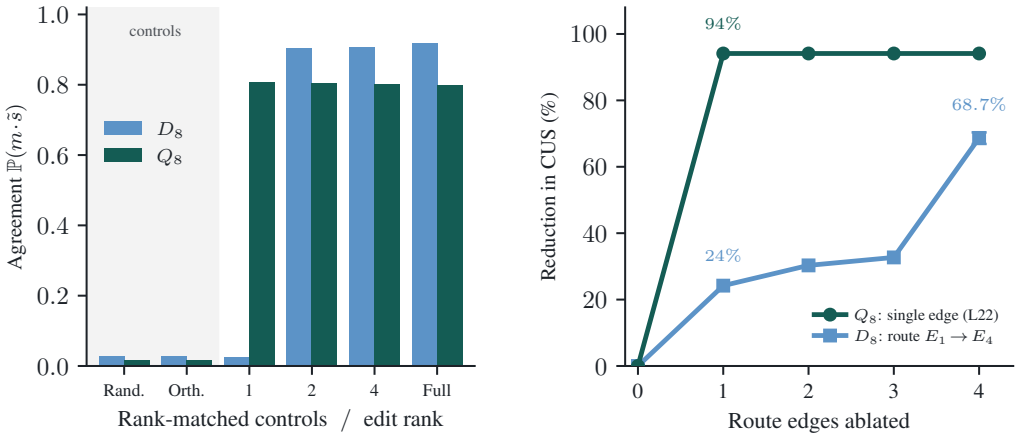
Edge	Alone	Leave-one-out drop
Layer 22, current hidden bit	0.212	0.346
Layer 19, move token	0.113	0.021
Layer 25, current hidden bit	0.078	0.039
Layer 23, current hidden bit	0.069	0.258
Full route E_4 (all four)	0.532	—

C.5 Register dimensionality and update-route concentration

These results separate register compactness from circuit concentration. Trace supervision creates an analogous compact register in both groups, and the route that updates it is concentrated in one edge in Q_8 and distributed across a small set of edges in D_8; the same token classes enter the nominated route in both groups. The main mechanistic contrast is the concentration of the update route.

The register itself is low-dimensional. Sweeping the edit rank shows the counterfactual update emerge and saturate by rank 1 in Q_8 and rank 2 in D_8 (Figure 11a), far below the 3584-dimensional residual stream. The rank profile is consistent with a one-bit register: in Q_8 a single causal direction suffices to hold and flip it. In D_8 the leading variance direction is not the causal one, so the rank-1 edit on its

own fails and even anti-selects the move-applied target, while its orthogonal complement still carries the effect; the second direction recovers it, placing the causal axis in a two-dimensional subspace.



(a) Compactness (b) Concentration

Figure 11: **A compact variable with a concentrated or distributed update route.** (a) Agreement is near zero for the random and orthogonal controls and reaches its plateau by rank 1 in Q_8 and rank 2 in D_8, far below the residual dimension, so the causal register is one to two dimensional; the D_8 rank-1 edit alone fails because its leading direction is misaligned with the causal one. (b) Cumulative reduction in selectivity as route edges are ablated one at a time. In Q_8 a single layer-22 edge already reaches the full 94 percent reduction, so the curve steps up at the first edge and saturates (concentrated). In D_8 no single edge suffices: the reduction accumulates along the frozen four-edge route (24, 30, 33 percent, on the selection split); the largest increment comes when the fourth edge is added, reaching a replicated partial 68.7 percent, the ten-split mean (distributed).

Layer and rank robustness. The register effect is stable across the trained layer and the neighboring layer, and a rank sweep shows the edit is already at full strength by rank 1 in Q_8 and rank 2 in D_8 (random and orthogonal controls stay near zero), so the register is one to two dimensional. The D_8 rank-1 edit is the lone exception: its leading direction is misaligned with the causal one, so it fails on its own while its orthogonal complement still carries the effect. The localization holds at the reported layer on both data splits (Table 6).

D STATE EDITING BEYOND THE GROUP TASKS

The extensions ask whether the state-use criterion continues to hold when several variables interact and when the edited value changes program control flow. We distinguish the original one-step and candidate-scoring experiments from the later generated-continuation experiments. A correct preference among supplied continuations does not establish that the model can generate the complete execution.

D.1 REPRODUCTION AND THE FIRST TASK EXTENSIONS

We first evaluated the original running-state checkpoints through the benchmark using the execution path described above. Matched rank-16 one-step counterfactual accuracy was 73.00% on Q_8 and 95.25% on D_8 for Qwen2.5-Coder-7B (400 examples per task), compared with values of 79.75% and 91.25%. Mistral-7B-v0.3 reached 100% on each task (2,000 examples each), compared with 92.65% and 94.30%. All four estimates met the prespecified absolute tolerance of ten percentage points. This is reproduction within a tolerance, not evidence of statistical equality. Mistral used fresh replication items; Qwen's item disjointness from the paper is not established.

The initial expanded-state study used two to four active fields, giving 8, 32, and 128 possible states, with entire rule systems held out. Across 64 easy, 128 medium, and 128 hard items, matched edits

gave 960/960 correct non-null one-step outcomes, 320/320 correct null outcomes, and 1,150/1,152 exact candidate-scored trajectories. The last denominator counts repeated item–horizon observations. Full replacement, visible rewriting, mean differences, and probe directions tied these scores. In a separate 16-case generation probe, all 64 requested updates were correct under matched edits, but every response continued past the requested length. Stopping generation after the requested number of lines gave 16/16 exact continuations.

The first interpreter study likewise separated candidate scoring from generation. Matched exact candidate trajectories were 32/32, 61/64, and 50/64 on easy, medium, and hard programs; branch scores were 32/32, 63/64, and 56/64. Yet only 4/16 medium and 0/16 hard generated continuations were exact. A line-count boundary improved formatting without improving execution. These results motivated evaluating complete generated continuations after verifying that the model could execute both original and visibly edited inputs. They are separate experiments, not additional observations in the final study below.

D.2 Tasks, splits, and generation

Factored state machines. The state contains a location L, mode M, and counter C, with $L, C \in \{0, 1, 2, 3\}$ and $M \in \{0, 1\}$. A rule card specifies the actions. Some actions change L only when M or C has a specified value. We edit that field to activate the condition. A paired action instead changes L in the opposite direction under the same condition. The 128 base edits contain 16 cases from each of eight unseen rule systems, balanced between mode and counter edits. These systems recombine the action schemas used in training; they do not introduce new operators. All edits activate an initially inactive condition, so the reverse direction is not tested. The final generated task has 32 possible states, rather than the 128 states in the largest earlier candidate-scoring task.

The primary condition places four unrelated updates before the consequential action. We also test the alternate action at this delay, the primary action after zero or eight updates, and an overwrite before use. Paired prompts share the prefix through the intervention token and place the future actions after it. Causal attention therefore gives the same activation at that token for both action sequences. The paired test asks whether this same edited value supports two different appropriate consequences.

Tiny interpreter. The machine has a program counter, two registers with arithmetic modulo eight, an equality flag, and a stack of at most four values. Instructions include arithmetic, comparisons, conditional jumps, push/pop, and halt. The 128 base programs contain 32 edits each to `r0`, `r1`, `flag_z`, and the top stack cell, with 64 edits in each direction of branch change. Each program is evaluated at first-use positions $d = 1, 4, 8$, after $d - 1$ unrelated arithmetic instructions. A flag is read directly by the branch, a register requires a comparison, and a stack value requires a pop and comparison. The branch therefore occurs at emitted step d, $d + 1$, or $d + 2$, respectively. This varies delay to one consequential branch; it does not increase the number of dependent branches. An overwrite condition removes the original/edited difference before use.

Evaluation protocol. Each task uses 256 calibration pairs, 64 development items, and 128 fresh final base items. Factored rule systems are disjoint across these splits and from continuation training. The current state is supplied at evaluation. After one interior activation edit, the model generates greedily without further intervention. No candidate future states are supplied. The simulator is used only for scoring. Factored generation stops at the publicly requested number of state lines. Interpreter generation stops at observable halt completion or end of sequence, with a common 2,048-token ceiling; the simulator's path length does not determine when to stop.

D.3 Training and the intervention

The final models are task-specific running-state adapters of `Qwen3-8B-Base`. Both original adapters failed a development requirement of at least 80% exact continuation on both clean and visibly rewritten inputs (Table 10). Each received one additional epoch on 2,048 original/edited prompt pairs (4,096 training rows), with 256 validation rows and correct stopping boundaries. The training data contain easy and medium cases and exclude final items and longest-delay cases. The edited value is supplied in text during this training. These are continued checkpoints, not the original adapters.

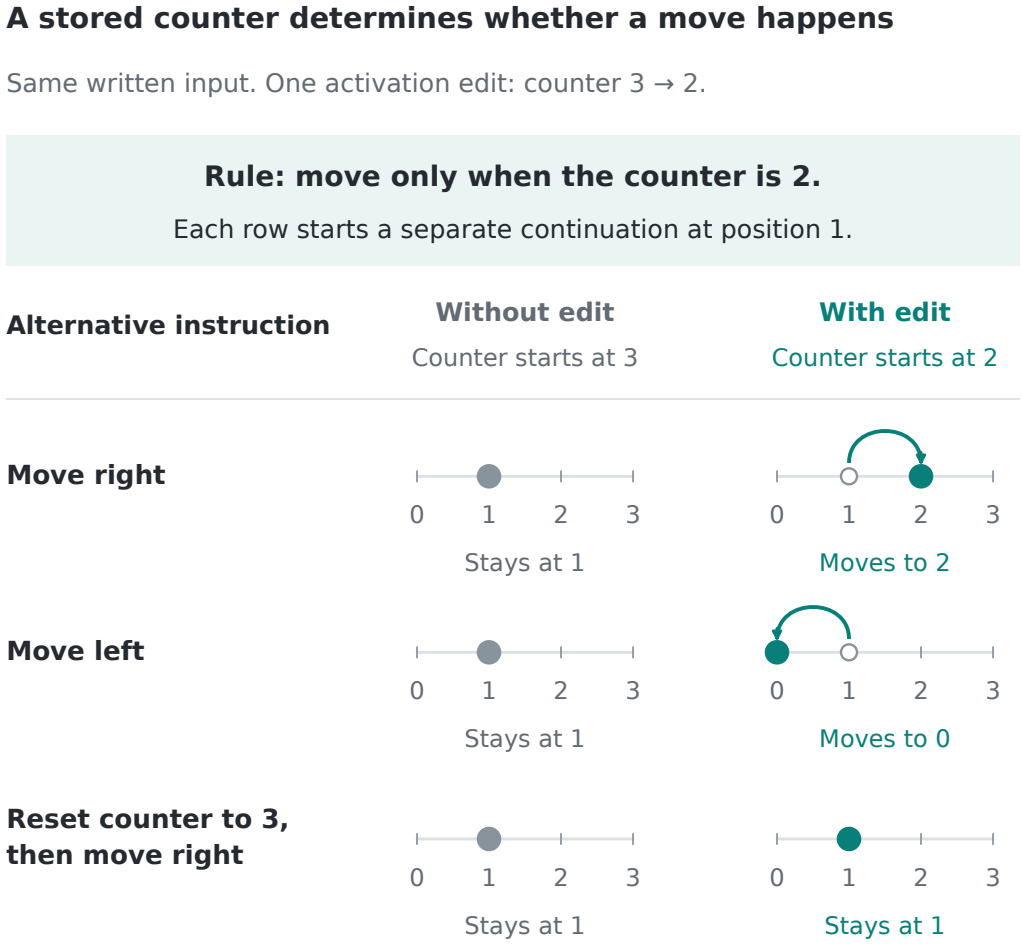


Figure 12: **Editing one state component changes how another updates.** The task has 32 possible states: four positions, two modes, and four counter values. Here an activation edit changes the counter used by the model from 3 to 2, while the written input stays unchanged. The illustrated movement rule then allows either a rightward or a leftward move, according to the later instruction. Resetting the counter removes this effect. Each row is a separate continuation, with four intervening updates before the counter is used. Position denotes L, counter denotes C, and the omitted mode ends at $M = 0$. The future instructions follow the edited token in the prompt.

Development checkpoint	Clean exact	Rewrite exact
Original factored, seed 0	44/64	60/64
Continued factored, seed 0	64/64	64/64
Continued factored, seed 1	64/64	64/64
Original interpreter, seed 0	22/64	19/64
Continued interpreter, seed 0	64/64	63/64

Table 10: **Development execution before final evaluation.** Before/after scores use the same development cases and are not independent test results.

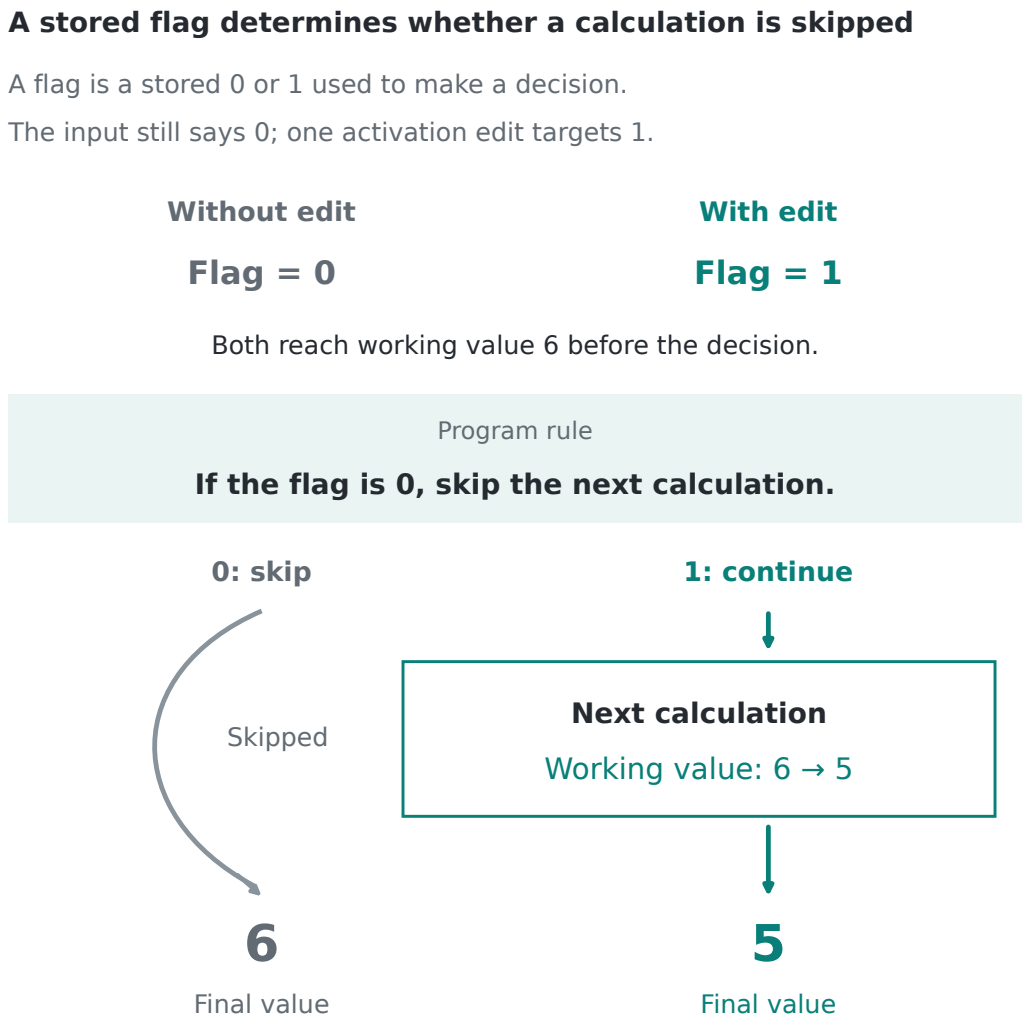


Figure 13: **An edited flag changes which calculation executes.** Both runs receive the same program and supplied state. Three arithmetic instructions bring the working register `r1` to 6 before a conditional jump reads the flag. Without editing, the flag is 0 and the next calculation is skipped. One activation edit targeting flag value 1 makes that calculation execute, so the final value is 5. The extra calculation adds seven modulo eight, giving $6 + 7 \equiv 5 \pmod 8$. The diagram expresses the branch rule in ordinary language and summarizes the generated traces. Other state fields are omitted; the full experiment also edits registers and the top stack value.

A second factored adapter was trained independently for four epochs on the original 16,192 training rows (1,592 validation rows), then received the same continuation procedure. The task for replication was completed before final evaluation. The two seeds share final items and rule systems; only training is replicated. The interpreter has one training seed.

Development compared full transplants at layers 10, 15, and 20. All tied on the downstream consequence, and the fixed tie-break selected layer 15. Edit strengths $0.5, 1, 2$ were compared for matched and mean-difference edits; strengths 1 and 2 tied, and the smaller was selected. Calibration, layer, strength, and decoding were frozen before final evaluation, independently for each trained adapter.

Let z and z^{donor} denote residual activations at the selected value token in prompts differing only in that value. For field f, the columns of U_f are the leading 16 right singular vectors of the uncentered calibration donor-minus-recipient differences. The matched intervention replaces the component in this subspace:

$$\widetilde{z} = z + U_f U_f^{\mathsf{T}} \left(z^{\text{donor}} - z \right).$$

This uses a paired donor activation, rather than the class-mean update in the original group assay. Comparators are no edit, visible rewriting, full donor replacement, a value-conditioned calibration mean difference $z + \mu_f(b) - \mu_f(a)$, and a random direction matched to the projected edit's norm. Here a, b are the original and edited values. The mean-difference comparator uses residual activations associated with known state values. It is distinct from the attention-head Function Vector construction examined elsewhere in the paper.

D.4 Generated consequences and complete execution

The *consequence* score checks the affected location in the factored task or the edited-state branch decision in the interpreter. *Exact continuation* requires every emitted state and its format to match the full reference execution. *Joint action success* requires both paired actions to succeed on the same factored edit. These metrics answer different questions: a correct branch need not imply a correct complete state or trace.

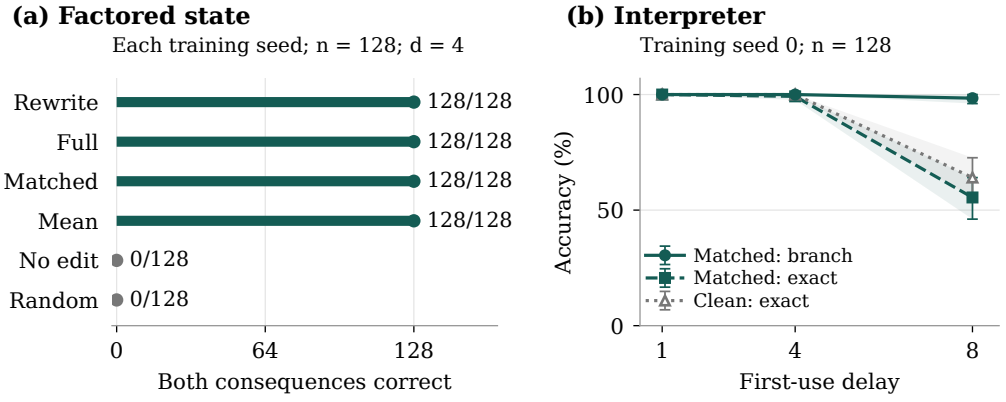


Figure 14: **Causal consequences and complete execution.** (a) Both alternative actions succeed after four intervening updates on the same 128 edits in each training seed; scores use the edited-state target. (b) Interpreter branch accuracy and exact continuation separate at longer delays. Delay is the first-use position, after zero, three, or seven unrelated instructions. The clean reference uses original-state truth; shading shows 95% program-bootstrap intervals. All interpreter methods are reported in Table 11.

Factored results. For each of the two seeds, matched rank-16, mean-difference, full replacement, and visible-rewrite interventions each achieve 128/128 correct consequences and exact continuations in every non-erasure condition. Joint success for the two delay-four actions is 128/128, and all unaffected fields are preserved. No edit and random directions achieve 0/128 against the changed

consequences while generating 128/128 exact *original* continuations. Thus, the zero counterfactual scores do not indicate failed clean execution.

One retained example starts from $L = 1, M = 1, C = 3$ and edits the counter to 2. Four intervening updates leave the counter unused. An action that increments the location when $C = 2$ then gives $(L, M, C) = (2, 0, 2)$; its paired decrement gives $(0, 0, 2)$. Without editing, both give $(1, 0, 3)$. Resetting the counter to 3 before the increment makes both final states $(1, 0, 3)$. These are observed generated outcomes, reproduced in both seeds, rather than a constructed illustration.

Interpreter results. Table 11 separates branch success from exact execution for all six methods and all three delays. Matched edits produce the correct branch on 128/128 programs at first-use positions one and four, with 128/128 and 127/128 exact continuations, respectively. At position eight, the requested value is delivered on 128/128 cases and retained through the dormant interval on 127/128. Branch accuracy is 126/128, while 71/128 full continuations are exact. Mean differences tie exact continuation and give 125/128 correct branches at this longest delay.

	Correct branch / exact continuation (each out of 128)		
Method	$d = 1$	$d = 4$	$d = 8$
No edit	0 / 0	0 / 0	0 / 0
Visible rewrite	128 / 128	128 / 127	125 / 75
Full replacement	128 / 128	128 / 127	125 / 72
Matched rank 16	128 / 128	128 / 127	126 / 71
Mean difference	128 / 128	128 / 127	125 / 71
Norm-matched random	0 / 0	0 / 0	0 / 0
No edit, original truth	128 / 128	128 / 128	128 / 82
Random, original truth	128 / 128	128 / 128	128 / 83

Table 11: **Interpreter branch use and execution accuracy.** The first six rows use counterfactual truth. The last two explicitly use original truth. Programs are shared across methods and delays; these are not independent cohorts.

Execution errors and reference conditions. At delay four, the same single program fails exact continuation under each of the four effective edit/rewrite methods: an intermediate flag is wrong even though the branch and final state are correct. The larger gap at delay eight separates correct use of the edited value from reliable execution of every state update.

All final interpreter outputs satisfy the format and public stopping requirements. The long-delay failures are therefore execution errors, rather than malformed or truncated responses. For example, in a retained failed case the edited register survives, but an arithmetic error is followed by an incorrect comparison and branch. Visible rewriting gets this branch right while still failing the full trace. Across the cohort, clean execution is exact on 82/128 and visible rewriting on 75/128, so the errors do not isolate damage caused by the activation edit.

For matched edits at delay eight, branch/exact counts by edited field are 31/10 for `r0`, 32/14 for `r1`, 31/15 for the flag, and 32/32 for the stack, each out of 32. The two branch-change directions give 63/29 and 63/42, each out of 64. These descriptive breakdowns reuse the same programs; they are not additional test sets.

Overwrite controls. After overwrite, all six methods reach the correct common downstream consequence and final state on 128/128 cases in both tasks and both factored seeds. The four effective edit/rewrite methods generate 128/128 exact counterfactual continuations. No edit and random directions instead generate 128/128 exact original continuations: their early states lack the edit even though their later outcomes coincide. Convergence after overwrite is the predicted null outcome, rather than a failed intervention.

Uncertainty and method comparisons. Reported 95% intervals use 2,000 bootstrap resamples of whole rule systems for the factored task or whole programs for the interpreter. Paired methods and repeated conditions remain together; training seeds are reported separately. Matched delay-eight

intervals are [96.1, 100]% for branch accuracy and [46.1, 64.1]% for exact continuation. Its one-program branch advantage over mean difference has a paired interval of [0, 2.34] percentage points, while exact continuation ties at every delay. There is no established matched-method superiority. Constant outcomes yield degenerate empirical bootstrap intervals; they do not establish universal reliability or population equivalence.

D.5 Original-adapter comparisons

We also compare the original running-state and final-answer adapters on a common one-update generation task. These adapters did not receive the continuation training above. Each was calibrated separately using 256 pairs at the shared frozen layer 15, rank 16, and strength 1; this is a common-site diagnostic, not a best-site search for each adapter.

	Factored		Interpreter	
Method	Running-state	Final-answer	Running-state	Final-answer
No edit (original truth)	95	10	128	74
Visible rewrite	125	126	113	65
Full replacement	125	126	111	63
Matched rank 16	125	126	111	62
Mean difference	123	126	121	64
Norm-matched random	0	0	0	0

Table 12: **Exact next-state counts for the original adapters.** Every entry is out of 128. Only the no-edit row uses original truth; all other rows use counterfactual truth. These scores do not measure later branches or complete trajectories.

The final-answer factored adapter's 126/128 rewritten next states coexist with only 10/128 correct clean states. It incorrectly increments the location on 118/128 clean cases, where the condition is inactive. Activating that condition makes the same location update appropriate. High rewritten accuracy alone therefore does not establish competence at following the conditional rule. On the original running-state interpreter, mean differences give 121/128 exact next states, compared with 111/128 for matched edits. After continuation training, the two methods achieve the same exact-continuation scores. Poor clean or rewritten execution also prevents interpreting an adapter's lower edited score as evidence that it lacks editable state.

Limited layer-zero full-replacement references give 32/32 correct consequences and exact continuations in each family, with the same generated text as layer 15 on those cases. The factored references cover only two of the eight held-out rule systems. These comparisons do not identify a uniquely effective interior layer or a minimal register dimension.

D.6 Scope and limitations

The supported extension is that an activation edit can supply a value used under held-out rule systems and in delayed program decisions during generated execution. It does not localize the update circuit in these tasks. The current state is supplied, and the generated text can preserve the edited value; these tests do not establish storage independent of the trace or replace the group experiments on model-generated prefixes. The backbone also differs from the original group experiments, so the scores do not isolate the effect of task complexity. The added continuation training and capability differences do not isolate the effect of supervision type. Finally, successful residual mean-difference edits here are compatible with the paper's Function Vector findings: state use is demonstrated by the downstream tests, not inferred from the fact that an average activation direction changes behavior.

E Function Vector construction and target-activation comparisons

The state-tracking comparison asks whether Function Vector steering directly writes the identified register. The lexical experiments ask whether effective steering recreates the matched target activation

at the intervention site. The constructions for these two settings are reported separately below, followed by the lexical scoring protocol and complete comparison results.

E.1 Function Vector construction for Q_8

We applied Function Vector steering to the Q_8-adapted `Qwen2.5-Coder-7B`. Vectors extracted from demonstration prompts transferred poorly to native Q_8 trace continuations, so we estimated head activations directly from those traces. At the current phase token, we estimated each attention head's mean output after the W^O projection from $1{,}024$ Q_8 trace examples, with 128 examples per state.

On 256 separate, state-balanced examples, we ranked heads by the average increase in the probability of the correct next-phase token when the head's target-minus-source mean was added at layer 12. The readout was conditioned on the correct next-region prefix. We selected the five heads with the largest positive average effects and used the same head set for every state.

For each region, we summed the selected heads' mean outputs separately for phases `P` and `R`, then subtracted the `P` sum from the `R` sum to obtain the regional steering vector. To change `P` to `R`, we added eight times this vector to layer-12 `resid_pre` at the current phase token; for the reverse edit, we subtracted the same scaled vector. The head count and intervention scale were selected on 256 development examples and frozen before evaluation on 400 held-out examples. Mean estimation, head ranking, development, and held-out evaluation used disjoint trace records.

E.2 Lexical tasks, vector construction, and comparison interventions

The steering experiments use `Qwen/Qwen2.5-7B` and `meta-llama/Llama-3.1-8B` in BF16 without quantization or chat templates. Task pairs and the Function Vector construction follow Todd et al. (2024). Ten demonstrations are rendered as `Q: input` and `A: output`; the query ends immediately after the `A:` prefix. Demonstration outputs are either correct or deterministically shuffled while preserving token alignment at the final answer position.

Inputs and outputs are split without replacement into 45 prompts for constructing the steering direction, 32 prompts for calibration, and 128 prompts for the initial evaluation. The comparison with the model's target activations uses 64 additional prompts per selected model-task setting. These prompts do not overlap with the construction, calibration, or initial evaluation data. Each target and shuffled prompt uses the same ten demonstrations and query.

Attention heads are ranked by average indirect effect. For each selected head, we replace its input to the output projection at the final prompt token with the corresponding average from target-task prompts while the model receives a shuffled prompt. The outcome is the probability of the first target token. The Qwen direction combines heads $(17, 15)$, $(18, 23)$, and $(16, 13)$; the Llama direction uses head $(13, 27)$. Their projected outputs are averaged over the 45 construction prompts. Separate calibration data select one intervention layer and coefficient for each model, yielding block 7 and coefficient 15 for Qwen, and block 8 and coefficient 10 for Llama.

The final comparison uses one effective setting per model:

Model	Task	Intervention block	Coefficient
Qwen2.5-7B	present-to-past	7	15
Llama-3.1-8B	English-to-Spanish	8	10

Each evaluation prompt has a deterministic alternative answer from the same output set, with the target-first and alternative-first order balanced 32/32. We score complete target-plus-EOS and alternative-plus-EOS log probabilities under the same prefix. In the all-layer target condition, we replace the activation at the answer position after every remaining block with the matched activation from the target run. In the one-layer target condition, we make this replacement only after the intervention block. Additive conditions use the Function Vector, the same direction at the norm of the prompt's natural activation change, the prompt's natural target direction at the Function Vector norm, a held-out average of 45 target-minus-shuffled activation changes, the other task's Function Vector, or a deterministic random direction.

E.3 SCORING AND NORMALIZATION

For each evaluation prompt i, we fix a target answer and an alternative answer and score the log probabilities of their complete strings, including the end-of-sequence token. Under condition c, the target-minus-alternative margin is

$$M_i(c) = \log p_c\big(y_i^{\text{target}} \mid x_i\big) - \log p_c\big(y_i^{\text{alternative}} \mid x_i\big),$$

where each answer log probability is the sum of its autoregressive token log probabilities. The paired gain from intervention c, relative to the same unedited shuffled prompt, is

$$\Delta_i(c) = M_i(c) - M_i(\text{unedited}).$$

To compare effects across models, we normalize by the repeated correct-prompt patch, abbreviated as the *all-layer patch*. This reference replaces the final-token hidden vector with its matched correct-prompt value after every remaining block. Its prompt-level gain is

$$G_i = M_i(\text{all-layer patch}) - M_i(\text{unedited}),$$

and the normalized effect is

$$E(c) = \frac{\frac{1}{n}\sum_{i=1}^{n} \Delta_i(c)}{\frac{1}{n}\sum_{i=1}^{n} G_i}.$$

Thus, $E(c) = 0$ indicates no average improvement over the unedited shuffled prompt, whereas $E(c) = 1$ indicates the same average margin gain as the all-layer patch. Negative values indicate movement toward the alternative answer. Values above 1 are possible because the all-layer patch is a reference rather than an upper bound. This score compares behavioral effects and does not measure similarity between the hidden vectors produced by different interventions.

We report paired 95% bootstrap intervals from 2,000 prompt-level resamples. Within each resample, we recompute both the intervention's mean paired gain and the all-layer reference gain, preserving the within-prompt comparison with the unedited condition. The all-layer patch has positive average gain in every bootstrap resample, so the normalization denominator is always well-defined. The task, model, intervention layer and coefficient, prompt sets, and scoring analysis were fixed before the outcomes in this comparison were examined.

E.4 TARGET-ACTIVATION PATCHES AND EQUAL-NORM COMPARISON RESULTS

Table 13: **Target-activation patches and comparison directions.** Entries are normalized effects with paired 95% bootstrap intervals. Patching the target activation through all later layers selects the target answer on every prompt in both models.

Condition	Qwen2.5-7B, present-to-past	Llama-3.1-8B, English-to-Spanish
Target activations at all later layers	1.000	1.000
Mean steering vector	0.544 [0.384, 0.720]	0.824 [0.596, 1.084]
Target activation at one layer	0.014 [0.005, 0.023]	-0.015 [-0.041, 0.005]
Natural target direction at steering norm	0.612 [0.391, 0.856]	-0.101 [-0.325, 0.094]
Mean steering at natural-change norm	0.0073 [-0.0003, 0.0155]	0.0166 [-0.0021, 0.0349]
Held-out average activation change	0.613 [0.332, 0.933]	0.085 [-0.085, 0.244]
Different-task steering vector	0.665 [0.505, 0.841]	-0.028 [-0.269, 0.192]
Random direction	0.175 [0.090, 0.270]	-0.316 [-0.501, -0.174]

The mean gains from patching target activations through all later layers are 4.412 nats [3.980, 4.853] for Qwen and 2.639 nats [2.206, 3.099] for Llama. The Function Vector norms are 210.80 and 20.22; the median natural activation-change norms are 2.74 and 0.78.

Validation of the all-layer reference. The all-layer patch serves as the positive reference used to normalize the effects of the other interventions. In each model-task setting, replacing the final-token hidden vector with its correct-prompt value after every remaining block causes the model to select the target answer on all 64 evaluation prompts. The patch also has positive average gain in every bootstrap resample, so the normalization denominator is always well-defined.

This result establishes that the correct-prompt hidden vectors contain sufficient information to recover the target behavior when they are repeatedly restored throughout the remaining computation. It does not show that the correct-prompt activation at any single block is sufficient, nor does it imply that the all-layer patch is an upper bound on the effect of other interventions. We therefore use it only as a successful causal reference against which to express the average behavioral gains of the local interventions.

Direction specificity. Because the calibrated steering vectors are much larger than the models' natural activation changes, we compare them with other directions rescaled to the same norm (Figure 3b). In Llama, the target task's steering vector has normalized effect 0.824. The prompt's natural target direction has effect −0.101, a steering vector for another task has effect −0.028, a held-out average activation change has effect 0.085, and a random direction has effect −0.316. Among the directions tested, the Llama effect is therefore specific to the constructed steering direction.

Qwen behaves differently. Its natural target direction, held-out average activation change, and steering vector for another task have effects 0.612, 0.613, and 0.665, compared with 0.544 for the target task's steering vector. A random direction is weaker at 0.175, so Qwen is not equally sensitive to every perturbation, but the target steering vector is not unique among structured directions of the same norm. Complete paired bootstrap intervals are reported in Appendix E.4.

Directional specificity strengthens the interpretation of Llama's behavioral response, but it does not establish state-like use. The one-block target-activation patch and natural-scale steering tests remain near zero, so even the direction-specific Llama effect does not show that the vector recreates the model's natural target activation at the tested layer.

F Steering persistence, output binding, and supplementary controls

These experiments test explanations of the steering effect beyond its immediate scale and direction: whether it becomes independent of the added vector, whether it follows reassigned output labels, and whether a measured local sensitivity accounts for it.

F.1 Removing the added vector at later layers

This experiment asks whether an additive steering edit is quickly converted into a self-sustaining internal change. We add the vector at its usual layer and then subtract the same vector at a later layer, leaving all other changes produced by the intervening computation intact. If the model has already converted the edit into another task-specific representation, the behavioral effect should remain after the original vector is removed.

For each model, we construct eight steering directions across four lexical tasks and data folds. Seven directions per model produce a clear steering effect relative to matched comparisons and are carried forward. In all fourteen cases, removing the vector eliminates the task-specific effect until late in the network. The first layers where much of the effect remains are blocks 22–24 in Qwen and 18–25 in Llama. Matched comparison directions also begin to affect the answer at blocks 9–15 and 8–10, respectively. Because task specificity and comparison-direction effects emerge at similar depths, the experiment does not isolate a task-specific representation that becomes self-sustaining early in the network.

F.2 Counterfactual output binding

We assign the correct answer to either of two arbitrary output labels while holding the underlying task fixed. For the paired assignments, let g_0 and g_1 be the edit-induced changes in the physical label-0-minus-label-1 margin. We decompose

$$S = (g_0 - g_1)/2, \qquad T = (g_0 + g_1)/2,$$

where S follows the semantic role assigned by the prompt and T remains tied to the physical labels.

Without steering, Qwen follows the assigned answer mapping on 93.3% of target pairs and Llama on 70.3%. Restricting the analysis to pairs solved under both mappings makes S more negative in both

Table 14: **The steering edit does not follow a counterfactual answer mapping.** The component that follows the assigned semantic role is negative in both models, while an effect tied to the physical output labels remains.

Model	Directions	S	95% CI	T
Qwen2.5-7B	7	-0.315	[-0.391, -0.236]	-1.094
Llama-3.1-8B	5	-0.250	[-0.278, -0.218]	0.225

models, so Llama's lower baseline agreement does not explain the sign of the main result. The vector therefore does not behave like an abstract "correct answer" signal that can be rebound to whichever output label the prompt designates as correct.

F.3 Supplementary local-sensitivity comparison

This supplementary local-sensitivity test is inconclusive because BF16 conversion misses the pre-specified tolerance by approximately 10^{-4}; we report it for completeness but do not use it to support the main claim.

We compare the Function Vector with a nearly orthogonal control direction constructed to have the same measured first-order effect on the target-minus-alternative margin. We also project the Function Vector to remove that measured first-order component and apply the unchanged vector only after the final decoder block.

Table 15: **Supplementary local-sensitivity comparison.** Effects are normalized to the paired gain from patching target activations through all later layers. The matching check described below misses its specified tolerance, so these values are included for completeness and are not used to support the main conclusion.

Condition	Qwen2.5-7B	Llama-3.1-8B
Mean steering vector	0.544 [0.390, 0.714]	0.824 [0.593, 1.072]
Sensitivity-matched control	0.097 [-0.090, 0.286]	-0.137 [-0.346, 0.062]
Sensitivity-removed steering vector	0.545 [0.391, 0.726]	0.796 [0.578, 1.056]
Steering only after the final block	-0.032 [-0.089, 0.026]	0.057 [-0.008, 0.126]

The near-orthogonality check is satisfied on all 128 prompts. After BF16 conversion, however, the relative norm-matching errors reach 9.81×10^{-5} in Qwen and 1.24×10^{-4} in Llama, above the specified 10^{-5} tolerance. We do not relax this criterion or rerun the experiment. The observed pattern is consistent with the measured scalar derivative and an unchanged final-layer readout being insufficient explanations, but the failed numerical check prevents a stronger conclusion and the experiment does not identify the downstream computation.

G Interpretation, scope, and related work

The distinction between behavioral control and computational state determines how we interpret both sets of experiments. We collect that interpretation, its limits, and the connections to prior work here.

G.1 What the intervention evidence establishes

Registers and update routes. The identified register is low-dimensional, although the route that updates it need not be equally concentrated. The causal effect saturates at rank one in Q_8 and rank two in D_8. In Q_8, ablating a single attention edge removes about 94% of the effect, whereas ablating the identified four-edge route removes 68.7% in D_8. The dimensionality of a stored variable and the concentration of the computation that reads and updates it are therefore separate properties.

The counterfactual branch also persists during free-running generation, cleanly in D_8 and more partially in Q_8, and the core register result replicates in a second model family. The circuit analysis

Table 16: **What different intervention tests establish.** Changing an answer demonstrates control. Interpreting the edit as computational state additionally requires showing that later operations use the edited value as the proposed variable.

Interpretation	Test	What the result shows
Direct output effect	Apply the edit at the final output boundary.	The intervention directly changes the reported answer.
Downstream control	Apply the edit before later layers and test whether behavior changes.	Later computation responds to the intervention.
Computational state	Write a counterfactual value, vary the next operation, and test the resulting future state.	The model uses the edited value according to the proposed variable's update rule.

itself is established only in the primary model. Full free-running, replication, and localization results appear in Appendices B.2, B.3 and C.

Scale and direction of steering. Replacing the final-token residual once with the exact residual from the correct prompt produces less than 2% of the all-layer reference effect in either model. Rescaling the Function Vector on each prompt to the size of the natural correct-versus-shuffled change is similarly ineffective. At its calibrated scale, the vector is effective, but its norm is 77 times the median natural change in Qwen and 26 times that change in Llama.

Successful steering at these sites is therefore not well described as restoring the residual that the model reaches on the matched correct prompt. Instead, the intervention moves the shuffled-prompt residual by a large, externally imposed displacement. This is the narrow, measured sense in which the steered residual is unnatural: the comparison concerns the immediate displacement at the intervention site, not the likelihood of that point under the full activation distribution or the geometry of its subsequent trajectory.

The equal-norm controls show that scale alone does not determine the effect. In Llama, only the target Function Vector produces a clearly positive target-directed effect among the equally large directions tested. Qwen is less selective, responding to several structured directions but more weakly to a random direction. Direction can therefore matter, but this does not establish that the intervention recreates a natural target activation or behaves like a register write. Full numerical results appear in Appendices E.4 and F.

Persistence and output binding. The cancellation experiment asks whether the initial steering displacement is quickly converted into a separate, self-sustaining task state. We add the vector at its usual layer and subtract it at progressively later layers, leaving the intervening computation intact. Early subtraction eliminates the task-specific effect, even though several layers have already processed the edited residual. Much of the effect survives only when subtraction is delayed until late in the network. The experiment therefore shows that the imposed displacement remains causally important under this test, but it does not identify how later layers use that displacement.

The answer-remapping experiment rules out another simple account. When the underlying task is held fixed but the output labels are reassigned, part of the steering effect remains tied to the same literal output label rather than following whichever label the prompt designates as correct. The vector is therefore not simply supplying an abstract "correct answer" signal. Detailed cancellation and answer-remapping results appear in Appendix F.

G.2 Scope and limitations

The state-tracking experiments use explicit traces in controlled Q_8 and D_8 tasks rather than unrestricted natural-language reasoning. The circuit analysis is established only in the primary model, although the central intervention result replicates in a second model family.

The steering experiments examine one selected task and layer per model, both chosen because conventional steering was effective. Model, task, layer, and steering scale therefore vary together, so the specificity difference should not be treated as a general contrast between Qwen and Llama.

The one-block patch replaces only the final prompt-token residual, while earlier token activations remain those of the shuffled prompt. Its failure rules out this local restoration account, not a task state represented at another position or layer, distributed across several activations, or constructed later in the network. Our use of *unnatural* is similarly local: we compare the immediate displacement at the intervention site with the matched natural change, but do not estimate the full distribution of natural residuals or follow the steered residual geometrically through every remaining layer.

Finally, the cancellation experiment shows that subtracting the imposed vector continues to remove the measured effect until late in the network, but not how later layers use the perturbation. The supplementary local-sensitivity comparison also misses its prespecified numerical tolerance by approximately 10^{-4}. We report it for completeness but do not use it to support the main conclusion.

G.3 Extended related work

Activation steering and task vectors. Activation Addition, representation engineering, inference-time intervention, and Function Vectors show that adding directions to internal activations can control language-model behavior (Turner et al., 2023; Zou et al., 2023; Li et al., 2023b; Todd et al., 2024). Function Vectors identify influential attention heads and average their outputs to elicit a task. Recent work further shows that activations taken just before the model performs a behavior are particularly effective sources for steering vectors (Ye et al., 2026). We ask a different question: when such a vector works, has it recreated an activation that the model naturally computes and continues from? Comparing the steering vector with the model's own target activation separates behavioral control from this stronger interpretation.

Identifiability and reusable interfaces. Behaviorally equivalent steering vectors need not identify a unique internal representation (Venkatesh and Mahendran Kurapath, 2026). The Hidden APIs framework uses multiple possible future operations to discover internal interfaces that can be transplanted and reused without assuming a known latent label (Ma et al., 2026). Our register experiment uses a complementary source of evidence: the state variable and transition rule are known by construction, so each counterfactual edit has a different predicted consequence under each next move. The activation-steering experiments then show why changing the answer alone does not provide the same evidence.

Causal abstraction and activation patching. Distributed alignment search and interchange interventions test whether model variables implement a proposed high-level computation (Geiger et al., 2024; 2021). Activation patching and causal tracing localize information by exchanging activations between runs (Meng et al., 2022). We compare several interventions that can all change behavior but support different conclusions: replacing one activation, replacing the sequence of activations through later layers, adding a fixed direction, and writing a counterfactual register value. The comparison asks what later computation can do with the edited activation rather than treating every successful patch as the same kind of explanation.

Scratchpads, state tracking, and world models. Scratchpads and chain-of-thought can improve multi-step behavior (Nye et al., 2021; Wei et al., 2022), but emitted text need not function as a variable in the model's later computation. Finite-state analyses identify decodable states in prompted Transformers (Zhang et al., 2025), while entity and belief-tracking work localizes mechanisms that bind and retrieve state from context (Prakash et al., 2024; 2025). Emergent world-state representations have also been found in self-supervised sequence models (Li et al., 2023a; Nanda et al., 2023b). We intervene on the supervised trace token and ask whether the model runs the known update rule on the value written there.

Geometry and causal use. Linear and nonlinear feature geometry, cyclic representations, and manifold steering provide useful descriptions of model representations and behavior (Engels et al., 2024; Nanda et al., 2023a; Kantamneni and Tegmark, 2025; Feucht et al., 2026; Wurgaft et al., 2026). Our result does not conflict with useful steering geometry. It asks for additional causal evidence before interpreting a direction as computational state: does the model's later computation use the edited value according to the proposed variable's semantics?