How Do Agentic LLMs Decide to Call Tools? A Tool-Call Vector Shaped by Suppression

Xijie Gong[1,2,*] **Tingxu Han**[1,3,*,†] **Jiahao Zhang**[1] **Wei Song**[4]
Ziqi Ding[5] **Hanqi Yan**[6] **Youcheng Sun**[1] **Lijie Hu**[1,‡]

[1]Mohamed bin Zayed University of Artificial Intelligence
[2]University of Electronic Science and Technology of China
[3]Nanjing University
[4]Griffith University
[5]University of New South Wales
[6]King's College London

Abstract

Tool calling, invoking external tools on demand, is central to agentic LLMs, yet the mechanism that decides whether a model calls a tool or responds directly remains poorly understood. Agentic prompts are long and heavily scaffolded, combining role instructions, tool schemas, format templates, and the user's request across hundreds of tokens, creating a noisy, highly entangled context in which no single controllable variable for mechanistic analysis is obvious. To obtain such a variable, we propose a method that converts complex agentic prompts into minimal contrastive pairs in which a single request verb determines the tool-call decision: replacing an execution-verb (e.g., *write*) with an analysis-verb (e.g., *discuss*) reliably flips the decision, suggesting it is mediated by a compact internal state. We construct 500 such paired prompts across Python, Java, and C++ (300 for mechanistic analysis, 200 held out for evaluation). We trace the decision to a vector, μ_Δ, that is both causally necessary and sufficient and generalizes beyond the discovery prompts to native multi-turn τ^2-Bench trajectories and verb-free requests. Behavioral ablations show that the scaffold establishes a tool-call prior; Transcoder decomposition then reveals that analysis verbs suppress this prior through features signaling that tool use is unnecessary, whereas execution verbs largely leave it intact. Downstream scaffold-reading attention heads and MLP features read out the resulting state, and the same mechanism recurs across seven models from the Qwen, Mistral, and Granite families. Our code is available at https://github.com/XijieGo/MI4ToolCalling.

1 Introduction

AI agents, from personal AI assistants to software engineering agents, are now widely deployed across industry and research. These agents are powered by agentic LLMs whose core capability is tool calling: invoking external tools on demand to act in the world. Given an agentic prompt containing a tool-call prompt scaffold (role instructions, tool schemas, and format templates) alongside the user's request, the LLM can generate a structured tool call specifying the selected tool and its arguments, enabling it to search the web, execute code [1, 2]. For any agentic prompt, the LLM faces the decision of whether to call a tool or respond directly. We call this the *call-or-no-call* decision. An incorrect decision causes the model to call tools unnecessarily or fail to call a tool when one is needed.

*Equal contribution †Project lead ‡Corresponding author. Lijie Hu <lijie.hu@mbzuai.ac.ae>

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

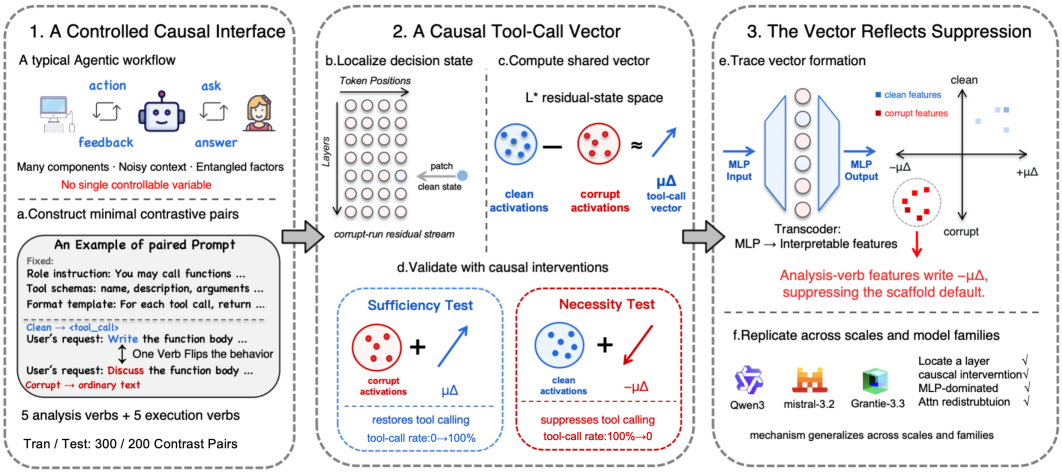


Figure 1: **Roadmap of the paper's mechanistic analysis.** The three numbered blocks summarize the stages of our analysis: (1) verb substitution turns long agentic prompts into a controlled causal interface; (2) the call-or-no-call decision passes through a tool-call vector, μ_Δ, that is causally sufficient and necessary; and (3) this vector reflects suppression of a scaffold-induced tool-call default and generalizes across the model families. The six lettered steps show how the evidence is built.

Frontier LLMs with dedicated agentic training often signal this call-or-no-call decision through a special token. In Qwen3, for instance, `<tool_call>` appears as the first generated token precisely when the model decides to call a tool [3]. In this setting, the call-or-no-call decision reduces to a simple question: is `<tool_call>` the model's top-1 prediction for the first generated token? This makes the call-or-no-call decision tractable for mechanistic analysis: a single-token output prediction is the natural target for mechanistic tools such as activation patching and direct logit attribution.

Yet what drives this decision has not been studied mechanistically. Existing work on tool calling has focused on whether LLMs select the right tools and how to improve their accuracy [1, 4–7]. Prior mechanistic work has typically examined short prompts of 10–30 tokens, where a single variable can be varied while holding the rest fixed [8, 9]. Agentic prompts are fundamentally different: the prompt scaffold (role instructions, tool schemas, and format templates) is packed alongside the user's request across several hundred tokens. With so many components that could each plausibly influence the decision, it is not clear which single element to vary to construct a reliable contrastive pair [10, 11]. This complexity makes it hard to isolate a single controllable variable for mechanistic analysis.

However, in code completion tasks, we find that the request verb serves as a controllable variable: replacing an execution verb with an analysis verb switches the first generated token in the retained pairs. Matched prompts differ only in the request verb, preserving the prompt scaffold, task body, and available tool. We construct 500 prompt pairs for each of the seven primary models, with 300 used for mechanistic analysis and 200 held out for evaluation. The coding tasks span Python, Java, and C++, drawing from MBPP, APPS, HumanEval, and CodeContests [12–15]. By holding the surrounding prompt fixed while varying a single behaviorally decisive token, these pairs isolate a controlled interface for mechanistic analysis.

We study this mechanism primarily in Qwen3-8B and test whether it generalizes across model scales and families. Activation patching localizes the decisive residual-stream state to layer 24: patching it recovers `<tool_call>` on 100% of corrupt prompts. Averaging the activation difference at this layer gives the tool-call vector, μ_Δ. Adding μ_Δ to analysis-verb prompts restores `<tool_call>`; subtracting it from execution-verb prompts suppresses tool calling. Because these pairs are behaviorally filtered to exhibit the intended contrast, we test whether μ_Δ generalizes beyond this construction. Furthermore, μ_Δ induces and suppresses tool calls in web retrieval, SQL execution, and email dispatch with new tool schemas and request verbs. The same control extends to native decisions in multi-turn, multi-tool τ^2-Bench trajectories across seven models, using model-specific μ_Δ without re-estimation. Even for requests that imply a need for tools without containing the execution or analysis verbs used in our prompt pairs, subtracting μ_Δ still suppresses tool calls. **The call-or-no-call decision passes through a compact tool-call vector shared across prompts.**

The natural hypothesis is that execution verbs actively generate a tool-call signal. Our evidence points to a different picture. The prompt scaffold establishes a tool-call prior: the role instructions, tool schemas, and format templates bias the model toward predicting `<tool_call>` first on task-bearing requests rather than empty or unrelated turns. Analysis verbs activate a family of features that signal the tool is unnecessary; they write against this prior through layers 21–23. Execution verbs activate these features less strongly and largely preserve the prior. μ_Δ is the inverse direction of the suppression signal that these features write into the residual stream. Downstream, scaffold-reading heads shift attention toward the tool-format template and MLP blocks support the opening of a tool-call sequence; both effects weaken when the suppression features are active. The mechanism replicates across the Qwen3-family and additional models, and structurally mirrors refusal [16]: a strong prior is present, and a compact suppressor overrides it. A scaffold-component ablation confirms this pattern behaviorally: the format template alone installs the near-ceiling call prior, and the tool schema makes it sensitive to wording.

To summarize, we make four contributions: **(1)** A first-generated-token target and controlled prompt pairs provide a clean causal interface for studying the call-or-no-call decision in long, scaffolded prompts (Section 2). **(2)** A tool-call vector μ_Δ at the layer-24 prediction position is causally necessary and sufficient for the call-or-no-call decision, and this control extends unchanged to real multi-turn agent trajectories and to requests with no cued verb at all (Section 3, Section 4). **(3)** μ_Δ is the inverse of the suppression signal that features activated by analysis-verbs write against a scaffold-induced, request-sensitive call prior; a scaffold-component ablation clarifies this prior behaviorally (Section 5). **(4)** Downstream attention heads and MLP blocks read this vector into the first generated token, and the same mechanism extends across Qwen3 and additional models (Section 6, Section 7).

2 One Verb Flips the Call-or-No-Call Decision

Mechanistic analysis requires a controlled change that produces a reliable behavioral contrast. Agentic prompts make such changes difficult to isolate. Role instructions, tool schemas, formatting rules, and user requests can all affect the decision. We seek a minimal change that exposes the call decision while preserving the task and scaffold.

We find a surprisingly simple control variable in code-completion tasks: a single request verb can reliably switch the model between calling a tool and responding directly. Asking the model to *write* a function often triggers a tool call, while replacing *write* with *discuss* can instead produce a text response (Figure 2). We therefore construct matched code-completion prompts that differ only in the request verb. Execution-verbs request an action on the code, and analysis-verbs request a textual response, while the task content, available tool, and scaffold remain fixed.

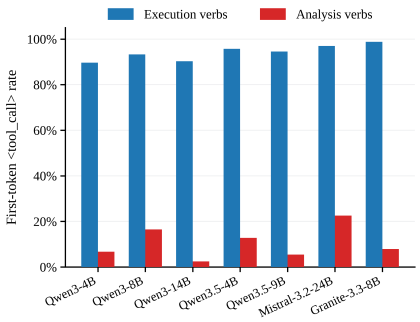


Figure 2: **Request verbs change tool-call rates across models.** Bars show first-token call rates on HumanEval before behavioral filtering, by request type.

We study Qwen3-8B [3] in non-thinking mode, where the first generated token exposes call initiation. A tool call begins with `<tool_call>`, and a direct response begins with ordinary text. We use five execution verbs (add, build, ...) and five analysis verbs (discuss, explore, ...). Tasks come from MBPP, APPS, HumanEval, and CodeContests [12–15] and cover Python, Java, and C++. The Qwen3-8B dataset contains 500 pairs, with 300 for mechanistic analysis and 200 held out for evaluation. Appendix A summarizes the data sources, splits, and prompt format.

> **Key Setup: A One-Word Causal Interface**
>
> Changing the request verb flips the first output between `<tool_call>` and direct text. The matched prompts isolate the call decision while preserving the surrounding task and scaffold.

Following activation-patching terminology, we call execution prompts *clean*, denoted x^c, and analysis prompts *corrupt*, denoted x^*. Each prompt has the form

$$x = \underbrace{[\text{role instructions},\ \text{tool schemas},\ \text{format templates}]}_{\text{prompt scaffold}}\ \underbrace{[v \parallel \text{task body}]}_{\text{user's request}} \longrightarrow y, \qquad (1)$$

Here v is the request verb, y is the first generated token, and $y_{\text{call}} =$ `<tool_call>`. The paired contrast is $y = y_{\text{call}}$ on execution prompts and $y \neq y_{\text{call}}$ on analysis prompts. Only v changes within a pair. We write $h_p^{(l)}(x) \in \mathbb{R}^{d_{\text{model}}}$ for the residual activation at layer l and position p.

3 The Tool-Call Vector

A shared residual vector controls the call-or-no-call decision in Qwen3-8B. We identify the vector in two steps. Activation patching localizes a *tool-call state* at the final prompt position. The mean execution–analysis difference at that state gives the *tool-call vector*.

3.1 Localizing the Tool-Call State

Activation patching locates the request verb's effect on the call decision [17, 18]. For each held-out pair (x_i^c, x_i^*), we patch the analysis state at layer l and position q with its execution counterpart. Other positions at that layer stay fixed; the rest of the model resumes from the patched state.

$$\tilde{h}_j^{(l,q)}(x_i^*) = \begin{cases} h_q^{(l)}(x_i^c), & j = q, \\ h_j^{(l)}(x_i^*), & j \neq q. \end{cases} \tag{2}$$

The recovery rate $r(l, q)$ is the fraction of patched prompts whose first-token top-1 prediction is y_{call}. We sweep layers at the verb position and the *prediction position*, the final prompt position that predicts the first output token. The patching effect shifts from the verb to the prediction position across layers. At L24, patching the prediction-position state recovers calls on 100% of held-out analysis prompts. This state is where we add and remove the tool-call vector.

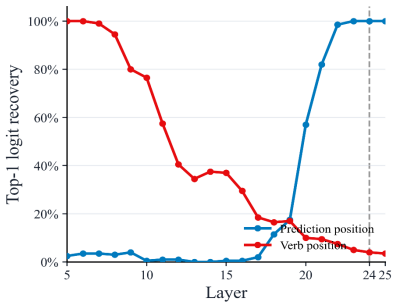


Figure 3: **The call signal shifts from the verb to the prediction position.** Layerwise patching finds the L24 residual state that determines when to call a tool or to give a text reply.

3.2 Estimating the Tool-Call Vector

We estimate the tool-call vector by averaging the execution-minus-analysis activation difference over the 300 training pairs at the L24 prediction position p.

$$\Delta_i = h_p^{(24)}(x_i^c) - h_p^{(24)}(x_i^*), \qquad \mu_\Delta = \operatorname*{mean}_i \Delta_i. \tag{3}$$

We then add μ_Δ to analysis-prompt activations and subtract μ_Δ from execution-prompt activations.

$$\text{Add:} \quad \tilde{h}_p^{(24)}(x_i^*) = h_p^{(24)}(x_i^*) + \mu_\Delta, \qquad \text{Remove:} \quad \tilde{h}_p^{(24)}(x_i^c) = h_p^{(24)}(x_i^c) - \mu_\Delta. \tag{4}$$

Both interventions change a single residual activation before downstream computation resumes. We measure each intervention's effect relative to the original execution–analysis logit gap. Let $\bar{z}_{\text{call}}^c$ and $\bar{z}_{\text{call}}^*$ denote mean tool-call logits on held-out execution and analysis prompts. Superscripts $* + \mu$ and $c - \mu$ denote addition and removal. The normalized effects are

$$\text{Suff}(\mu_\Delta) = \frac{\bar{z}_{\text{call}}^{*+\mu} - \bar{z}_{\text{call}}^*}{\bar{z}_{\text{call}}^c - \bar{z}_{\text{call}}^*}, \qquad \text{Necc}(\mu_\Delta) = \frac{\bar{z}_{\text{call}}^c - \bar{z}_{\text{call}}^{c-\mu}}{\bar{z}_{\text{call}}^c - \bar{z}_{\text{call}}^*}. \tag{5}$$

Addition and removal recover most of the logit gap in their respective directions (Table 1). The same vector therefore supports both tool-call induction and suppression at the prediction position.

Table 1: **The tool-call vector causally controls the call-or-no-call decision.** Adding μ_Δ to analysis prompts induces tool calls, while removing it from execution prompts suppresses them.

Intervention	Top-1 before	Top-1 after	Δ top-1	Logit before	Logit after	Δ logit	Suff.	Necc.
Add μ_Δ	0.00%	100.00%	↑100.00	25.11	32.50	↑7.39	1.03	–
Remove μ_Δ	100.00%	0.00%	↓100.00	32.28	24.80	↓7.48	–	1.04

Circuit searches identify relevant components, but high-fidelity replay requires distributed computation (Appendix G). The vector gives us a compact causal target for tracing how those components form and read out the call decision at the first generated token.

4 Generalization Beyond Verb-Cued Coding

The Qwen3-8B discovery prompts combine a coding task, an explicit request verb, and a single-turn scaffold. We test how the tool-call vector behaves as these conditions change. The tests cover new domains and wording, native multi-turn trajectories, and implicit requests. Every Qwen3-8B intervention keeps the coding-derived direction fixed at the L24 prediction position; cross-domain experiments calibrate its norm, while τ^2 and verb-free tests apply fixed gains to the original vector.

4.1 Transfer Across Tools, Domains, and Action Wording

We test transfer to web retrieval, read-only SQL execution, and email dispatch using 100 held-out call/no-call pairs per domain. The prompts retain the discovery scaffold and introduce new task content and tool schemas. Call-side verbs such as *retrieve*, *execute*, and *dispatch* fall outside the coding action vocabulary. Within each pair, only the leading request token changes. We keep the coding-derived direction fixed at the L24 prediction position and rescale the vector to match the call–no-call contrast norm estimated from separate target-domain training pairs.

Vector addition induces tool calls on all corrupt prompts, and vector removal suppresses tool calls on clean prompts (Table 2). The normalized logit shift averages 0.88 for addition and 0.94 for removal. A norm-matched random vector changes no top-1 decisions. The coding-derived vector supports bidirectional control of the first-token call decision across tools and action vocabularies.

Table 2: **The coding-derived vector controls tool calling across web retrieval, SQL execution, and email dispatch.** Qwen3-8B interventions keep the coding-derived direction fixed at L24 and match the vector norm to each target domain. N counts held-out pairs. Induction and suppression rates measure first-token switches from no-call to call under addition and from call to no-call under removal, respectively. Norm. shift is the mean `<tool_call>` logit increase for Add or decrease for Remove, divided by the baseline call–no-call logit gap. Means weight domains equally.

			Add μ_Δ		Remove μ_Δ	
Target	Tool	N	Induction (%)	Norm. shift	Suppression (%)	Norm. shift
Web retrieval	`web_search`	100	100.0	0.93	100.0	0.75
SQL execution	`run_sql`	100	100.0	0.79	100.0	0.77
Email dispatch	`send_email`	100	100.0	0.91	100.0	1.30
Mean	—	300	**100.0**	**0.88**	**100.0**	**0.94**

4.2 Native Multi-Turn, Multi-Tool Trajectories

We next apply the coding-derived vector, without re-estimation, inside ongoing agent interactions. We evaluate native τ^2-Bench Telecom trajectories [19] with 5,385–15,690 context tokens, 16–43 available tools, and prior interaction and call history. We intervene at the next decision point without changing the request, using 200 decision points in each of the removal and induction arms.

At $1\times$ gain, vector removal suppresses 36.7% of baseline tool calls, and vector addition induces tool calls on 70.0% of turns with a baseline text response (Table 3). A norm-matched random vector switches the call decision on 1.5% and 0.0% of these turns, respectively. Across the 200 evaluated decision points in each arm, removal decreases the mean `<tool_call>` logit by 5.55, while addition increases it by 19.42. These paired logit changes capture a shift in call preference beyond the decisions whose top-1 token flips. The coding-derived vector therefore remains effective under long contexts, multiple tools, and the accumulated history of interactions and tool calls.

4.3 Verb-Free Requests

Requests can also imply a need for tool use without an explicit instruction to act. We construct 600 non-imperative requests across code, search, database, and API tasks. The requests include questions such as *Is it true that . . . ?* and statements such as *I haven't gotten to . . . yet*. We call these requests *verb-free* because they contain no explicit action verb specifying what the model should do.

Table 3: **The coding-derived Qwen3-8B vector transfers beyond the discovery prompts.** All entries use $1\times$ gain at L24. N counts evaluated decision points. Top-1 flip is the fraction of baseline calls suppressed by removal or baseline text responses changed to calls by addition. Δz_{call} is the mean paired after-minus-before change in `<tool_call>` logit over all N points. Top-1 rate gives the call rate before and after intervention over the same N points. Appendix C gives random controls and shows how the verb-free results vary with the chosen intervention gain.

Setting	Intervention	N	Top-1 flip (%)	Mean Δz_{call}	Top-1 rate (before $\rightarrow$ after)
τ^2-Bench, native call	Remove	200	36.7	-5.55	99.5% $\rightarrow$ 63.0%
τ^2-Bench, native text	Add	200	70.0	$+19.42$	0.0% $\rightarrow$ 70.0%
Verb-free, native call	Remove	160	93.1	-9.95	100.0% $\rightarrow$ 6.9%

Among 160 evaluated requests with a baseline Qwen3-8B tool call, vector removal suppresses 93.1% of calls at $1\times$ gain. At $1.5\times$ gain, the suppression rate reaches 100.0%, compared with 16.2% for a norm-matched random vector at the same gain. The paired logit changes in Table 3 show a decrease in tool-call preference. This supports reuse of the call-state direction on requests that omit the explicit verb cues used to discover μ_Δ. Appendix C.2 gives the request construction and selection procedure.

Finding: A General Tool-Call Vector Within the Model

Using controlled verb contrasts and activation patching, we identify a tool-call vector μ_Δ within the model. Intervening along this direction controls the call-or-no-call decision across tools and domains, including native multi-turn trajectories and implicit requests.

5 How Does the Tool-Call Vector Form?

The vector's transfer raises a mechanistic question. How do the scaffold and request create the state captured by μ_Δ? We return to the controlled Qwen3-8B setting, establish the scaffold's behavioral effect, and trace the vector's formation to individual layers and features.

5.1 The Scaffold Establishes a Tool-Call Prior

The scaffold creates a strong tendency to call tools on task-bearing requests. On held-out tasks, neutral requests produce a mean call probability of 0.8486, while analysis requests reduce the probability to 0.0038. We separate the scaffold into role instructions (R), tool schemas (T), and format templates (F) to identify the source of this tendency. The format template establishes the call prior, while the tool schema makes the resulting call tendency sensitive to the request (Table 4). Removing F nearly eliminates calling. With F alone, call probabilities approach one for both request types. Removing T weakens the neutral–analysis separation. A length-matched replacement for T produces an intermediate separation, indicating that both schema content and context length contribute.

The prior is gated by the presence of task-relevant content. Empty turns and unrelated questions elicit no top-1 calls. The task body alone yields a 32.3% call rate. Execution requests reach 100%, while analysis requests reduce the rate to 0% in this baseline experiment. We therefore use *scaffold-induced call prior* to denote this task-conditioned tendency to call tools.

Table 4: **The scaffold creates a request-sensitive call prior.** Entries are mean first-token call probabilities on 200 held-out Qwen3-8B tasks. Appendix D.1 gives all component combinations and a control that replaces the tool schema with text of the same length.

Scaffold	Neutral request	Analysis request
$R + T + F$ (full)	0.8486	0.0038
$R + T$ (no format template)	6.32×10^{-7}	8.31×10^{-10}
$R + F$ (no tool schema)	0.9521	0.4108
F only	1.0000	0.9952

5.2 MLPs Dominate Tool-Call Vector Formation

We track the call signal by projecting residual activations onto the unit direction $\hat{\mu}_\Delta = \mu_\Delta / \|\mu_\Delta\|$. At the prediction position, the mean execution–analysis gap at layer l is

$$g_l = \frac{1}{N} \sum_{i=1}^{N} \langle h_p^{(l)}(x_i^c) - h_p^{(l)}(x_i^*), \hat{\mu}_\Delta \rangle. \tag{6}$$

We also project each MLP and attention-head output onto $\hat{\mu}_\Delta$ to measure component contributions.

The gap g_l stays near zero through the first fifteen layers, grows modestly in the middle layers, and rises sharply in L20–L23. Thus, we call L20–L23 the *formation window*. MLPs supply 77.7% of the projected write in this window, compared with 22.3% for attention heads (Figure 4). MLP23 contributes the largest single write, equal to 22.5 on the g_l scale.

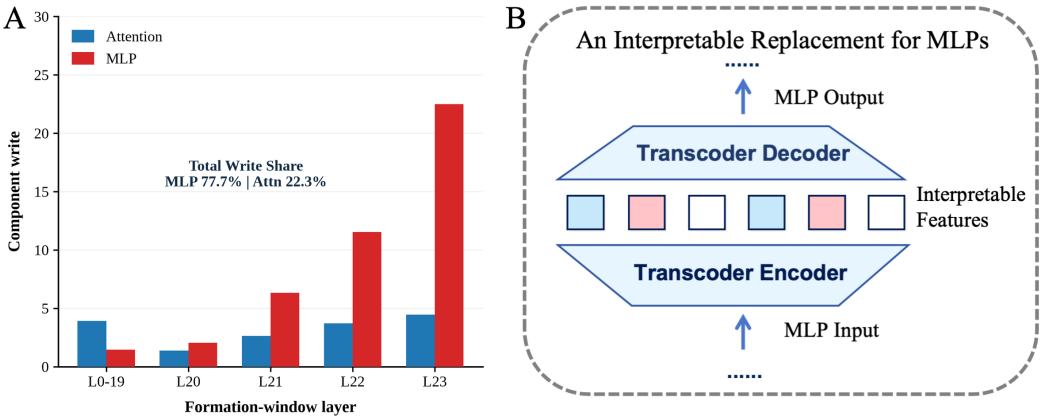


Figure 4: **MLPs dominate formation-window writes.** Left, Qwen3-8B component outputs projected onto $\hat{\mu}_\Delta$. Right, the Transcoder architecture used to decompose MLP writes.

5.3 Suppression Shapes the Vector in the Formation Window

We decompose formation-window MLP writes with Transcoders, which approximate each output with feature-weighted decoder directions [20]. Feature selection uses 300 training pairs; scores and interventions use 200 held-out pairs and the fixed L24 vector.

$$\mathrm{MLP}^{(\ell)}(x) \approx b_{\mathrm{dec}}^{(\ell)} + \textstyle\sum_f a_{\ell f}(x)\, w_f^{(\ell)}, \tag{7}$$

where $a_{\ell f}(x) \geq 0$ is the ReLU feature activation of the normalized MLP input at the prediction position, and $w_f^{(\ell)}$ is its decoder direction. The bias cancels in paired contrasts. We score each feature by its activation contrast and projection onto $\hat{\mu}_\Delta$,

$$\kappa_{\ell f} = (\bar{a}_{\ell f}^c - \bar{a}_{\ell f}^*) \langle w_f^{(\ell)}, \hat{\mu}_\Delta \rangle, \tag{8}$$

where $\bar{a}_{\ell f}^c$ and $\bar{a}_{\ell f}^*$ are mean activations on clean and corrupt prompts.

Positive κ increases the execution–analysis gap; negative κ reduces it. The gap can arise from execution-active features writing along $\hat{\mu}_\Delta$ or analysis-active features writing against it. We sum contribution magnitudes by activation side,

$$K_{\mathrm{corrupt}}^{(\ell)} = \sum_{f:\bar{a}_{\ell f}^* > \bar{a}_{\ell f}^c} |\kappa_{\ell f}|, \qquad K_{\mathrm{clean}}^{(\ell)} = \sum_{f:\bar{a}_{\ell f}^c > \bar{a}_{\ell f}^*} |\kappa_{\ell f}|. \tag{9}$$

Let $\beta_{\ell f} = \langle w_f^{(\ell)}, \hat{\mu}_\Delta \rangle$. The net contributions of suppressive and promoting writes are $S^{(\ell)} = \sum_{f:\beta_{\ell f}<0} \kappa_{\ell f}$ and $E^{(\ell)} = \sum_{f:\beta_{\ell f}>0} \kappa_{\ell f}$, respectively. Retaining both signs of κ gives

$$S^{(\ell)} - E^{(\ell)} = K_{\mathrm{corrupt}}^{(\ell)} - K_{\mathrm{clean}}^{(\ell)}. \tag{10}$$

This identity also holds after summing across layers, so $K_{\text{corrupt}}/K_{\text{clean}} > 1$ indicates a larger net suppressive contribution to the execution–analysis gap.

We label each layer's 80 largest positive and 80 largest negative training-κ features using max-activating prompts, activation patterns, and decoder tokens. Both the statistics for feature families and those for the full feature set use the same held-out κ values.

Table 5: **Features more active on analysis prompts dominate the formation window.** K_{corrupt} and K_{clean} sum $|\kappa|$ over all features with higher analysis and execution activations, respectively, on 200 held-out pairs. Share is each layer's fraction of the net Transcoder feature write over L20–L23. Appendix D.4 gives the statistics for the labelled families of Transcoder features.

Layer	Dominant	K_{corrupt}	K_{clean}	$K_{\text{corrupt}}/K_{\text{clean}}$	Share (%)	Semantic label
L20	Clean	0.24	0.93	0.26	4.6	Execution requests
L21	Corrupt	1.47	1.10	1.34	16.7	Non-necessity
L22	Corrupt	1.85	1.33	1.39	22.8	Analysis-task contexts
L23	Corrupt	5.98	1.43	4.18	55.9	Analysis-verbs

The scaffold already supports frequent calls on neutral requests (Section 5.1), providing a behavioral baseline for interpreting the execution–analysis contrast. Across the formation window, analysis-active features carry nearly twice the contribution magnitude of execution-active features (Table 5). In the *analysis-non-execution* family, which has the largest total contribution among the labelled families, most features are more active on analysis prompts and write against $\hat{\mu}_\Delta$. Their stronger analysis-side activation therefore enlarges the paired gap through stronger negative writes on analysis prompts. These observations support an asymmetric formation process: execution requests largely preserve the scaffold-induced call tendency, while analysis requests actively suppress it. The execution-minus-analysis vector thus largely reflects the suppression present on the analysis side.

We next examine whether this analysis-side dominance translates into substantial influence over the call decision. If the identified features actively restrain the call tendency, removing their writes should increase calling while the analysis request remains unchanged. Zeroing the writes of the five features with the largest suppressive contributions ($|\kappa|$), comprising less than 0.001% of the 655,360 Transcoder features across L20–L23, shifts the L24 state along $+\hat{\mu}_\Delta$ and raises the call rate on analysis prompts to 25.5% (Appendix D.4). The increase in calling under unchanged request wording supports a causal role for these features in suppressing the scaffold-induced call tendency.

> **Mechanistic Insight: Suppression Shapes the Tool-Call Vector**
>
> The scaffold establishes a conditional tool-call prior. Analysis requests activate suppressor features that write against the tool-call direction $\hat{\mu}_\Delta$, while execution requests largely preserve the prior. This asymmetry between request types shapes the tool-call vector.

6 How Does the Tool-Call Vector Control the Call Decision?

Downstream attention heads and MLP features read the L24 tool-call state into the first generated token. We trace this readout with direct logit attribution (DLA) and single-component patching over L25–L35. DLA measures each component's direct contribution to the tool-call logit [21].

6.1 Scaffold-Reading Heads Support Tool Calling

To identify candidate readers of the L24 call state, we compare downstream attention heads' direct logit attribution (DLA) to `<tool_call>` on execution and analysis prompts. L29H9, L33H11, and L33H29 combine stronger execution-side call-token support with a shared shift of attention from role instructions toward the tool-format template (Figure 5A–B). We therefore refer to them as *scaffold-reading heads*. We then test whether their call-directed writes respond to the internal state by adding μ_Δ at L24 while leaving the analysis prompt unchanged. This intervention brings all three heads' DLA close to their execution-side values (Table 24). The recovered output support under fixed prompt content shows that the internal call state regulates these heads' call-directed writes, supporting their role in reading available scaffold information into the call-opening token.

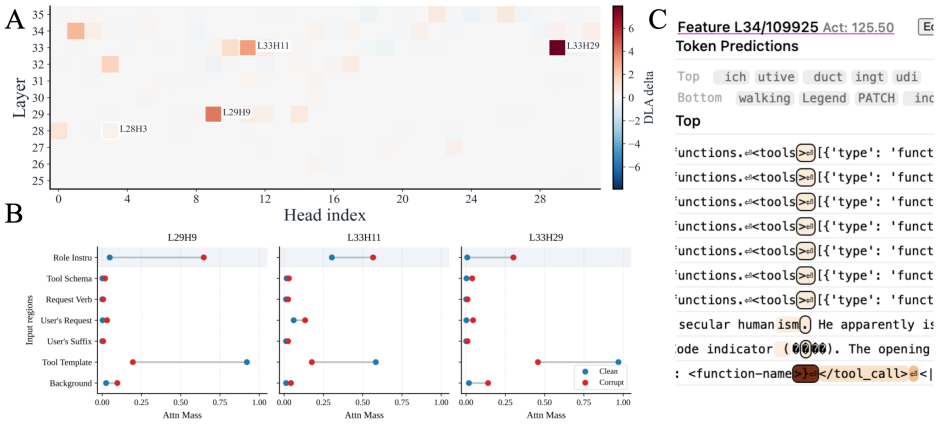


Figure 5: **Scaffold-reading heads and a late structural feature read out the tool-call state.** (A) Execution-minus-analysis DLA for the `<tool_call>` logit across Qwen3-8B attention heads in L25–L35. (B) L29H9, L33H11, and L33H29 share a shift toward the tool-format template and away from role instructions on execution prompts (blue) relative to analysis prompts (red). (C) Max-activating contexts for L34/F109925 highlight tool-schema boundaries, supporting its interpretation as a structural readout feature for the opening of a tool call.

6.2 Committing to the Tool-Call Opening

MLP34's strong patching effect motivates a feature-level test of how the call state is expressed in the output format. Transcoder feature F109925 responds to tool-schema boundary contexts and writes toward the call-opening token (Figure 5C). Adding μ_Δ at L24 brings both its activation and its call-token write on analysis prompts close to their execution-side values, linking the feature's activity to the internal call state. To test whether restoring this feature also changes the decision, we replace its analysis-side activation with the paired execution-side value at the prediction position. On the same 200 held-out pairs, this yields 37.0% calling, compared with 6.5% for the fixed non-structural control F91365 and 41.0% for the full MLP34 patch (Appendix E). Its response to μ_Δ and the behavioral recovery from feature replacement together support F109925 as a causal contributor to expressing the call state in the token that initiates tool use.

> **Takeaway: From Scaffold Prior to Tool-Call Output**
>
> The scaffold establishes a conditional tool-call prior. Analysis requests activate suppressor features that write against the tool-call direction $\hat{\mu}_\Delta$, while execution requests preserve the prior. Scaffold-reading heads and late structural MLP features then read the resulting state into the `<tool_call>` opening as the first token of the response.

7 The Tool-Call Vector Across Scales and Model Families

We ask whether the mechanism established in Qwen3-8B replicates across model scales and families, and whether the resulting vectors generalize beyond the discovery prompts: (i) the decision state localizes to a prediction-position layer $(l, r(l, p))$; (ii) μ_Δ passes causal intervention tests at that layer (Suff., Necc.); (iii) the vector generalizes beyond the controlled verb contrasts to new domains, native multi-turn τ^2-Bench trajectories, and verb-free requests (Domains, τ^2, Verb-free); (iv) formation-window writes are MLP-dominated and suppressor-driven (MLP/Attn, $K_{\text{corrupt}}/K_{\text{clean}}$); and (v) scaffold attention varies with the call state (Max Attn). We test Qwen3-4B and 14B, Qwen3.5-4B and 9B, Mistral-Small-3.2-24B, and Granite-3.3-8B at the fixed block-input layers in Table 6.

Across all seven models, localization and causal intervention remain consistently strong, while the strength of generalization and downstream readout varies across architectures. Full-state patching recovers calls on 93.5–100.0% of analysis prompts. Vector interventions yield normalized sufficiency scores of 0.81–1.03 and necessity scores of 0.61–1.04. More importantly, $K_{\text{corrupt}} > K_{\text{clean}}$ in every model, preserving the central asymmetry identified in Qwen3-8B: the tool-call vector is shaped primarily by suppressive computation on the analysis side rather than by an execution-specific signal.

Table 6: **The tool-call mechanism recurs across seven models and generalizes beyond the discovery prompts.** Domains and τ^2 average conditional intervention rates; Verb-free reports suppression at unit gain. Domains uses norm calibration (Appendix I.1). Max Attn summarizes clean-minus-corrupt scaffold-attention shifts (Appendix F.2). For hybrid Qwen3.5, MLP/Attn is omitted and attention is summarized only for layers with full attention.

	Localization		Intervention		Generalization (%)			Formation		Readout
Model	l	$r(l,p)$ (%)	Suff.	Necc.	Domains	τ^2	Verb-free	MLP/Attn	$K_{\text{corrupt}}/K_{\text{clean}}$	Max Attn (pp)
Qwen3-4B	26	100.0	0.97	0.71	87.7	72.2	100.0	2.08	3.26	82.6
Qwen3-8B	24	100.0	1.03	1.04	100.0	53.3	93.1	3.48	1.99	72.6
Qwen3-14B	34	100.0	0.97	0.84	100.0	85.0	100.0	6.87	2.41	87.3
Qwen3.5-4B	31	99.0	0.84	0.61	91.1	47.9	100.0	–	1.58	50.3
Qwen3.5-9B	31	100.0	0.92	0.88	100.0	80.2	100.0	–	1.30	52.2
Mistral-Small-3.2-24B	25	100.0	0.90	0.76	100.0	83.5	100.0	3.93	1.61	40.5
Granite-3.3-8B	35	93.5	0.81	0.72	43.0	70.3	35.5	7.52	2.11	36.5

Together, these results suggest that the specific layers and components vary across models, while the functional organization of the call decision remains stable across these models.

8 Related Work

Tool-use research develops and evaluates models that decide whether and how to invoke external tools [1, 4, 2, 6, 7]. These studies establish the call-or-no-call decision as a central agent capability. Its causal internal representation remains largely uncharacterized. Mechanistic interpretability localizes causal states and resolves component computations [17, 18, 9, 20], while representation engineering, function vectors, and refusal directions show that compact residual directions can mediate behavior [22, 23, 16]. We connect these lines of work by extending causal mechanistic analysis to a core agent behavior: deciding whether to initiate tool use. Controlled request contrasts isolate this decision in long, scaffolded prompts, revealing a localized tool-call vector that transfers across tools, domains, and request forms. Tracing its formation and downstream readout explains how request semantics suppress a scaffold-induced call prior, linking behavioral tool-use evaluation to an internal causal account of action initiation. Appendix H provides a review of related literature.

9 Conclusions

A single tool-call vector gives causal control over the call-or-no-call decision at the first output token in Qwen3-8B. The vector is estimated from one-word request contrasts and transfers without re-estimation to native multi-turn trajectories and verb-free requests. The mechanistic analysis explains how the controlled contrast arises. The scaffold establishes a call prior on task-bearing requests, analysis requests activate suppressor features, and execution requests largely preserve the prior. Downstream attention and MLP features read the resulting state into the tool-call opening. The vector connects these computations through a compact, reusable representation, providing a concrete account of how the model uses request semantics to regulate tool calling.

10 Limitations and Broader Impacts

Our scope is the mechanism of the binary call-or-no-call decision. The primary formation analysis uses behaviorally screened coding pairs under a fixed scaffold, so its applicability to other prompt constructions remains to be established. We do not evaluate the broader correctness of tool use, including whether a call selects an appropriate tool, produces valid arguments, or completes the underlying task. The formation analysis identifies a scaffold-induced call prior, a vector, and broad suppressive feature families. It does not yet resolve the fine-grained computation that maps scaffold information and request semantics to the vector. Transfer supports reuse of a downstream call state, but does not establish identical formation mechanisms for explicit and implicit requests. Our findings may help developers understand and regulate when AI agents use tools. The same interventions could also induce tool calls that conflict with a user's intent.

References

[1] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761. 1, 2, 10, 29

[2] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X. 1, 10, 29

[3] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388. 2, 3, 29

[4] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023. URL https://arxiv.org/abs/2305.15334. 2, 10, 29

[5] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789. 29

[6] Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Gong, and Lichao Sun. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/bc12914d66b41b6bfc2d3a5decdb498b-Abstract-Conference.html. 10, 29

[7] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/patil25a.html. 2, 10, 29

[8] Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. *arXiv preprint arXiv:2211.00593*, 2022. URL https://arxiv.org/abs/2211.00593. 2, 29

[9] Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16318–16352. Curran Associates, Inc., 2023. doi: 10.52202/075280-0719. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf. 2, 10, 27, 29

[10] Yi-Chang Chen, Po-Chun Hsu, Chan-Jan Hsu, and Da-shan Shiu. Enhancing function-calling capabilities in LLMs: Strategies for prompt formats, data integration, and multilingual translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume*

3: Industry Track), pages 99–111, 2025. doi: 10.18653/v1/2025.naacl-industry.9. URL https://aclanthology.org/2025.naacl-industry.9/. 2

[11] Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. AGENTIF: Benchmarking instruction following of large language models in agentic scenarios. *arXiv preprint arXiv:2505.16944*, 2025. URL https://arxiv.org/abs/2505.16944. 2, 29

[12] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732. 2, 3, 29

[13] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=sD93GOzH3i5. 19, 29

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374. 29

[15] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158. 2, 3, 29

[16] Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. *arXiv preprint arXiv:2406.11717*, 2024. URL https://arxiv.org/abs/2406.11717. 3, 10, 30

[17] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc., 2022. doi: 10.52202/068431-1262. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf. 4, 10, 29

[18] Stefan Heimersheim and Neel Nanda. How to use and interpret activation patching. *arXiv preprint arXiv:2404.15255*, 2024. URL https://arxiv.org/abs/2404.15255. 4, 10, 29

[19] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. URL https://arxiv.org/abs/2506.07982. 5, 19, 29

[20] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable LLM feature circuits. *arXiv preprint arXiv:2406.11944*, 2024. URL https://arxiv.org/abs/2406.11944. 7, 10, 30

[21] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html. 8, 29

[22] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to ai transparency, 2025. URL https://arxiv.org/abs/2310.01405. 10, 30

[23] Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron Wallace, and David Bau. Function vectors in large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 17282–17333, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4ae163cb8788970e53b4fd9578141139-Paper-Conference.pdf. 10, 30

[24] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1074. URL https://aclanthology.org/N18-1074/. 19

[25] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/. 19

[26] Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. *arXiv preprint arXiv:2403.17806*, 2024. URL https://arxiv.org/abs/2403.17806. 28

[27] Lin Zhang, Wenshuo Dong, Zhuoran Zhang, Shu Yang, Lijie Hu, Ninghao Liu, Pan Zhou, and Di Wang. EAP-GP: Mitigating saturation effect in gradient-based automated circuit identification. In *Advances in Neural Information Processing Systems*, volume 38, pages 157478–157504, 2025. doi: 10.52202/085713-4748. URL https://papers.nips.cc/paper_files/paper/2025/hash/d029c97ee0db162c60f2ebc9cb93387e-Abstract-Conference.html. 28, 29

[28] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. *arXiv preprint arXiv:2304.08244*, 2023. URL https://arxiv.org/abs/2304.08244. 29

[29] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045. 29

[30] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=RIu5lyNXjT. 29

[31] Ella Rabinovich and Ateret Anaby-Tavor. On the robustness of agentic function calling. In *Proceedings of the 5th Workshop on Trustworthy NLP*, pages 298–304, 2025. URL https://aclanthology.org/2025.trustnlp-main.20.pdf.

[32] Hongfei Xia et al. SafeToolBench: Pioneering a prospective benchmark to evaluating tool utilization safety in LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://aclanthology.org/2025.findings-emnlp.958/. 29

[33] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016. doi: 10.48550/arXiv.1610.01644. URL https://arxiv.org/abs/1610.01644. 29

[34] John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, 2019. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275/.

[35] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7/. 29

[36] Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023. URL https://arxiv.org/abs/2303.08112. 29

[37] Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. *arXiv preprint arXiv:2310.10348*, 2023. URL https://arxiv.org/abs/2310.10348. 29

[38] Lin Zhang, Lijie Hu, and Di Wang. Mechanistic unveiling of transformer circuits: Self-influence as a key to model reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1387–1404, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.76. URL https://aclanthology.org/2025.findings-naacl.76/. 29

[39] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models. *arXiv preprint arXiv:2304.14767*, 2023. URL https://arxiv.org/abs/2304.14767. 30

[40] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023. URL https://arxiv.org/abs/2309.08600. 30

[41] Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024. URL https://arxiv.org/abs/2406.04093.

[42] Tim Lawson et al. Residual stream analysis with multi-layer SAEs. In *NeurIPS 2024 InterpretableAI Workshop*, 2024. URL https://openreview.net/forum?id=vr5VRKq09l.

[43] Samuel Marks et al. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024. URL https://arxiv.org/abs/2403.19647. 30

[44] Michael Hanna and Emmanuel Ameisen. Latent planning emerges with scale, 2026. URL https://arxiv.org/abs/2604.12493. 30, 31

[45] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023. URL https://arxiv.org/abs/2308.10248. 30

[46] Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*, 2023. URL https://arxiv.org/abs/2312.06681. 30

[47] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *arXiv preprint arXiv:2306.03341*, 2023. URL https://arxiv.org/abs/2306.03341. 30

[48] Xinyan Jiang, Wenjing Yu, Di Wang, and Lijie Hu. Global evolutionary steering: Refining activation steering control via cross-layer consistency. *arXiv preprint arXiv:2603.12298*, 2026. URL https://arxiv.org/abs/2603.12298. 30

[49] Manjiang Yu, Hongji Li, Priyanka Singh, Xue Li, Di Wang, and Lijie Hu. PIXEL: Adaptive steering via position-wise injection with exact estimated levels under a subspace calibration. In *Proceedings of the ACM Web Conference 2026*, pages 1574–1585. ACM, 2026. doi: 10.1145/3774904.3792273. URL https://doi.org/10.1145/3774904.3792273. 30

[50] Manjiang Yu, Hongji Li, Junwei Chen, Xue Li, Priyanka Singh, Yang Cao, and Lijie Hu. Multi-adapter representation interventions via energy calibration. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 150526–150544. PMLR, 2026. URL https://proceedings.mlr.press/v306/yu26v.html. 30

A Experimental Setup and Dataset Construction

This appendix describes the model-specific paired-prompt datasets and the controls that support the Tool-Call Vector. Each model in the seven-model comparison has 500 clean/corrupt pairs. Within a pair, the task or conversation and tool interface stay fixed while the request verb changes. Call outcomes are measured at the first output token.

A.1 Task Sources for Each Model's Paired Prompts

The model-specific corpora use programming tasks from APPS, CodeContests, HumanEval, and MBPP, or conversation contexts from τ^2-Bench Telecom. Source families vary by model; Table 7 summarizes the assignment.

Table 7: Source families represented in the model-specific paired corpora.

Source family	Models	Task content
APPS, CodeContests, HumanEval, MBPP	Qwen3-4B, Qwen3-8B, Qwen3-14B, Qwen3.5-9B, Granite-3.3-8B	Programming tasks with statements, function signatures, examples, tests, or reference solutions.
τ^2-Bench Telecom	Qwen3.5-4B, Mistral-Small-3.2-24B	Conversation histories with a controlled request-verb contrast.

For coding tasks, normalization preserves the function behavior, programming language, and available examples or tests. The paired prompts keep each model's task context and tool interface fixed while varying the request verb.

A.2 Qwen3-8B Prompt Template

The Qwen3-8B discovery prompts use Qwen's chat template. The system turn provides the `write_file` tool and states the required `<tool_call>` serialization format. The assistant prefix uses the non-thinking completion form used by Qwen3. The boxed template below shows the structure; braces mark fields filled by each task. Cross-model corpora use each model's native prompt template and call-opening token.

Agentic prompt template for the controlled Qwen3 coding pairs.

```
<|im_start|>system
# Tools

You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type":"function","function":{"name":"write_file",
"description":"Write file.",
"parameters":{"type":"object","properties":{"file_path":{"type":"string"},
"content":{"type":"string"}},"required":["file_path","content"]}}}
</tools>

For each function call, return a json object with function name
and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call><|im_end|>
<|im_start|>user
{REQUEST_VERB} the function body in {TARGET_FILE} based on the function
definition and docstring below:
{TASK_BODY}
<|im_end|>
<|im_start|>assistant
<think>

</think>
```

The tool list, tool schema, output format instruction, target file, task body, and assistant prefix are identical within each pair; the request verb at the beginning of the user turn is the only change.

The call marker is `<tool_call>` for Qwen3 and Qwen3.5, `[TOOL_CALLS]` for Mistral, and `<|tool_call|>` for Granite. The evaluation compares the native token's logit with all competing first tokens. For inputs stored as native token IDs, evaluation uses those IDs directly.

A.3 Pair Selection and Data Splits

The paired corpora are organized separately by model. Pairs are retained for an execution-call/analysis-text behavioral contrast. Each model has 500 pairs, with the split and source family shown in Table 8. Verb pools are model-specific. Vector fitting uses training pairs only, and intervention results are evaluated on held-out pairs. The main paired-prompt experiments use 300 training pairs and 200 held-out pairs for every model. Additional appendix controls use the evaluation cohorts specified in their respective subsections.

Table 8: Per-model paired-prompt corpus sizes and splits.

Model	Source family	Train	Held-out
Qwen3-4B	Coding tasks	300	200
Qwen3-8B	Coding tasks	300	200
Qwen3-14B	Coding tasks	300	200
Qwen3.5-4B	τ^2-Bench Telecom	300	200
Qwen3.5-9B	Coding tasks	300	200
Mistral-Small-3.2-24B	τ^2-Bench Telecom	300	200
Granite-3.3-8B	Coding tasks	300	200

A.4 Qwen3-8B Dataset Composition

The Qwen3-8B corpus contains 500 pairs, split into 300 training and 200 held-out pairs. Its verb counts by split are given in Table 9; the source composition is given in Table 10.

Table 9: Qwen3-8B execution- and analysis-verb counts by split.

Split	Execution verbs	Analysis verbs
Train (300)	add 60, build 60, complete 60, save 60, write 60	discuss 71, explore 75, inspect 6, review 72, study 76
Held-out (200)	add 53, build 20, complete 20, save 53, write 54	discuss 40, explore 40, inspect 40, review 40, study 40

Table 10: Source composition of the Qwen3-8B paired-prompt corpus.

Source	Train	Held-out
APPS	192	51
CodeContests	63	135
HumanEval	15	4
MBPP	30	10
Total	300	200

A.5 A Representative Clean/Corrupt Prompt Pair

The example below instantiates the task-bearing user turn and assistant prefix of the template in Appendix A.2. The paired analysis request changes *Save* to *Explore*; the shared system turn and tool schema follow that template.

Representative execution request and completion prefix, following the shared system turn.

```
<|im_start|>user
Save the function body in solve.py based on the function definition and
docstring below:
def isLongPressedName(name: str, typed: str) -> bool:
    """
    Given strings name and typed, return whether typed could result from
    typing name where characters may be long-pressed and repeated one or
    more times.
    >>> isLongPressedName('alex', 'aaleex')
    True
    """
    pass
<|im_end|>
<|im_start|>assistant
<think>

</think>
```

Paired analysis request. All remaining prompt content is unchanged.

```
Explore the function body in solve.py based on the function definition and
docstring below:
```

B Identifying the Tool-Call Vector

The main text localizes the call-or-no-call state to the prediction position and summarizes it with the mean clean–corrupt residual difference μ_Δ. The controls below test whether the causal effect depends on the selected layer and direction. They use the same first-token call metric as the main experiments and separate the learned orientation from perturbation norm.

B.1 L24 Supports Both Call Induction and Suppression

The neighboring-layer control tests how the causal effect depends on the intervention layer. We fit a mean-difference direction at the input residual stream of L22, L23, L24, and L25 using 200 training pairs and evaluate each direction on 300 disjoint held-out pairs from a separate control corpus. This neighboring-layer control uses a 200/300 split; the main Qwen3-8B vector uses the 300/200 split above. Each learned vector is compared with an equal-norm random direction at the same layer. Matching layer and norm isolates the contribution of the learned orientation.

Table 11: Neighboring-layer control with independently fitted directions. Entries are held-out post-intervention call rates. Random controls match the learned direction's norm at each layer.

Intervention	L22	L23	L24	L25
Analysis $+\mu_\Delta$	82.0%	100.0%	100.0%	100.0%
Execution $-\mu_\Delta$	45.7%	12.0%	0.0%	0.0%
Analysis + random	0.0%	0.3%	18.7%	0.0%
Execution − random	100.0%	99.7%	97.7%	100.0%

The addition and removal arms reveal complementary aspects of call control. Induction becomes complete at L23, while complete suppression appears at L24. L24 is therefore the earliest tested layer that combines both effects, which also persist at L25 (Table 11). The weaker effects of norm-matched random directions link this control to the fitted orientation. Together with the full-state patching result, this makes L24 a well-supported reference state for tracing formation and downstream readout.

C Generalization and Transfer

The paired prompts hold each model's task context and tool interface fixed while varying the request verb. Transfer tests whether the resulting call-control direction remains effective across task domains, tool schemas, contexts, and request wording. Each model retains its fitted direction throughout; target-domain norm calibration adjusts its magnitude using separate training pairs.

C.1 Call Induction and Suppression in Native τ^2-Bench Trajectories

We evaluate τ^2-Bench [19] trajectories at each model's next native decision point: Telecom for the Qwen models and Retail for Mistral and Granite. Each arm begins with 200 candidate decision points. Removal is evaluated on the N_{call} points whose baseline first token is the model's native call marker; induction is evaluated on $N_{\text{text}} = 200$ baseline text decisions. We apply the fitted μ_Δ and a norm-matched random direction at unit gain. Tables 12 and 13 report strict first-token flip rates for the two arms; the removal table gives each model's baseline-call denominator.

Table 12: τ^2-Bench removal: strict first-token flip rate among the N_{call} native baseline tool-call decisions.

Model	N_{call}	μ_Δ, 1×	Random, 1×
Qwen3-4B	199	61.3%	0.5%
Qwen3-8B	199	36.7%	1.5%
Qwen3-14B	200	97.0%	8.0%
Qwen3.5-4B	198	88.4%	0.0%
Qwen3.5-9B	196	91.3%	17.3%
Mistral-Small-3.2-24B	200	75.0%	0.5%
Granite-3.3-8B	200	95.5%	2.0%

Table 13: τ^2-Bench induction: strict first-token flip rate among native text decisions.

Model	μ_Δ, 1×	Random, 1×
Qwen3-4B	83.0%	0.0%
Qwen3-8B	70.0%	0.0%
Qwen3-14B	73.0%	0.0%
Qwen3.5-4B	7.5%	0.0%
Qwen3.5-9B	69.0%	0.0%
Mistral-Small-3.2-24B	92.0%	0.0%
Granite-3.3-8B	45.0%	0.0%

Transfer in native trajectories tests whether the discovery direction controls call initiation as the surrounding task, history, and tool set change. The fitted directions outperform norm-matched random perturbations in both arms (Tables 12 and 13). Each intervention leaves the native request and conversation fixed, so the resulting flip connects the transferred internal state to the call decision in that interaction. Induction and removal establish this connection from both baseline text and baseline call states, extending the direction's action-initiation role to ongoing conversations with different tools and histories.

C.2 Call Suppression on Verb-Free Requests Across Models

We construct 600 verb-free requests from APPS [13], FEVER [24], Spider [25], and an API domain, using five non-imperative phrasing patterns with 30 requests per domain–pattern combination. Requests are rendered in each model's native template. After evaluating the baseline on all 600 requests, we select up to ten tool-call-positive requests per domain–pattern combination in source order. This subset is fixed across the removal and random-control interventions. We subtract μ_Δ at gains $\alpha \in \{1, 1.5\}$ and compare against a norm-matched random direction at $\alpha = 1.5$. Table 14 reports the strict drop rate of each model's native tool-call marker among these baseline-positive requests.

Table 14: Implicit, verb-free generalization across all seven models. N is the evaluated baseline-positive subset; r is a norm-matched random direction.

Model	N	$(-\mu_\Delta)$, $\alpha = 1$	$(-\mu_\Delta)$, $\alpha = 1.5$	$(-r)$, $\alpha = 1.5$
Qwen3-4B	117	100.0%	100.0%	34.2%
Qwen3-8B	160	93.1%	100.0%	16.2%
Qwen3-14B	161	100.0%	100.0%	44.7%
Qwen3.5-4B	186	100.0%	100.0%	33.9%
Qwen3.5-9B	187	100.0%	100.0%	73.8%
Mistral-Small-3.2-24B	147	100.0%	100.0%	5.4%
Granite-3.3-8B	200	35.5%	88.5%	27.5%

Removing the fitted direction suppresses calls even when requests omit the explicit verb cues used to discover it. The same direction therefore controls call initiation across explicit and implicit request forms (Table 14). The gain response further shows how this control depends on displacement along the direction: increasing the gain raises Granite's suppression from 35.5% to 88.5%. The learned-direction advantage over norm-matched random perturbations at the same gain connects this effect to the fitted orientation. These results support a reusable downstream call state reached through different request formulations.

D Formation of the Tool-Call Vector

The formation analysis asks how the scaffold and request wording produce the state identified in Appendix B. Scaffold and affordance controls establish when a call tendency is prepared and how requests regulate it. We then connect request information, intermediate call control, and output support, before testing the feature writes that form the call state.

D.1 The Scaffold Establishes a Task-Conditioned Call Prior

We ablate role instructions (R), tool schemas (T), and format templates (F) on the 200 held-out tasks in the balanced Qwen3-8B dataset. A length-matched control replaces the tool schema T, while retaining R and F. The neutral requests use a balanced assignment of *Consider*, *Handle*, *Take*, *Use*, and *Process*. Table 15 gives the component ablations and the length-matched control.

Table 15: Scaffold-component ablation, Qwen3-8B, 200 held-out prompts per condition. p_{call} is the mean first-token `<tool_call>` probability; P is the neutral-minus-analysis gap.

Scaffold components	Neutral p_{call}	Analysis p_{call}	P
$R + T + F$ (full scaffold)	0.8486	0.0038	0.845
$R + F$ (no tool schema)	0.9521	0.4108	0.541
$R + T$ (no format template)	6.32×10^{-7}	8.31×10^{-10}	6.3×10^{-7}
F only	1.0000	0.9952	0.005
T only	1.87×10^{-8}	2.05×10^{-11}	1.9×10^{-8}
R only	8.77×10^{-9}	7.14×10^{-11}	8.7×10^{-9}
Empty system scaffold	6.80×10^{-16}	1.05×10^{-15}	-3.7×10^{-16}
R + length-matched neutral text + F	0.7311	0.1553	0.576

The factorial ablations separate a high call tendency from sensitivity to the request. With F alone, both request types strongly favor the call opening; removing F nearly eliminates that tendency. With role instructions and the format template held fixed, including T strengthens the neutral–analysis separation. Replacing T with length-matched neutral text weakens this separation even though context length is retained, supporting a role for schema content beyond token count (Table 15). The residual separation under replacement shows that the retained scaffold and request wording also contribute to request sensitivity.

The separate 300-task request cohort shows that the scaffold establishes a call tendency conditional on task context (Table 16). Neutral wording already elicits frequent calls; execution wording reinforces this tendency, while analysis wording suppresses it. Empty, greeting, and unrelated turns keep calling

inactive. These controls place the asymmetry in how the request regulates a task-conditioned prior: execution requests preserve or strengthen a prepared action, and analysis requests override it.

Table 16: Qwen3-8B request baselines on a 300-task evaluation cohort. Empty, greeting, and unrelated turns are evaluated once each. The full scaffold is present except in the final row.

Request	Call top-1	Mean call probability
Empty turn, greeting, or unrelated question	0.0%	$\approx 5.0 \times 10^{-4}$ (max.)
Task body only	32.3%	0.3310
Neutral verb and task body	86.7%	0.8547
Analysis verb and task body	0.0%	0.0080
Execution verb and task body	100.0%	0.9993
Execution verb and task body, no scaffold	0.0%	6.56×10^{-17}

D.2 Tool Affordances Change Whether a Request Elicits a Call

The available tool changes what it means to act on a request. We cross *Write* and *Review* with `write_file` and `submit_review`, using 100 requests per condition. The *Review* request elicits 4.0% calling with the writing tool and 100.0% when review submission is available (Table 17). Review becomes an actionable instruction when the interface provides the corresponding operation. The execution/analysis contrast therefore depends on the relation between request meaning and tool affordances, which gives the discovery verb pools their behavioral significance in that interface.

Table 17: Request–tool crossover on Qwen3-8B. Entries are tool call rates, with 100 requests.

Request verb	`write_file`	`submit_review`
Write	100.0%	34.0%
Review	4.0%	100.0%

D.3 Request Encoding Precedes Formation and Readout of the Call State

We ask how the timing of request encoding relates to formation of the call state and its expression in the output token. At the prediction position, a linear probe measures whether the residual stream distinguishes execution from analysis requests, the logit lens measures the call-token preference, and MLP DLA identifies direct writes toward that token. The probe first exceeds 0.9 AUC at L1, the call-token logit-lens gap first exceeds 1.0 at L27, and the largest MLP DLA differences occur at L33–L35 (Table 18). Thus, the request distinction is available early, while its strongest direct expression in call-token writes occurs near the output.

The formation and intervention results explain the role of the computation between these timing landmarks. Formation-window MLPs build the execution–analysis gap along $\hat{\mu}_\Delta$ in L20–L23 (Section 5.2). At L24, adding μ_Δ on analysis prompts induces calling, and subtracting it on execution prompts suppresses calling, with the request held fixed in each arm (Section 3). Changing this intermediate state therefore redirects the eventual decision before the lens gap crosses 1.0 and before the strongest MLP call-token writes. Together, these results place call-state formation between early access to the request distinction and the late output writes: the request contrast is already encoded when the formation-window computation produces a state that can regulate the subsequent decision.

Table 18: Held-out early-signal diagnostics and layer-wise gap accumulation on the 200 held-out pairs (single-sample TransformerLens protocol with standard single RMSNorm).

Diagnostic	Result
Probe AUC > 0.9 first layer	L1
Logit-lens gap > 1 first layer	L27
DLA top-3 layers (MLP write)	L35, L34, L33
Final gap mean (single RMSNorm, 95% CI)	7.093 [6.687, 7.498]
First 10% / 25% / 50% of final gap	L25 / L29 / L29
Largest increments	L29, L33, L24, L27, L32

D.4 Analysis-Active Feature Writes Suppress the Call Tendency

The main text defines the signed feature contribution κ and the two activation-side contribution masses. To interpret the writes forming the L24 state, we select the 80 largest positive and 80 largest negative training-κ candidates per layer across L20–L23. This gives 640 distinct layer–feature candidates from the 655,360 features in the formation window. Selection and family assignment use the 300 training pairs; contribution magnitudes are evaluated on the 200 held-out pairs.

Family assignment combines decoder-token cues with the request contexts that activate each feature. We inspect the 20 highest- and 20 lowest-projection tokens under each decoder and the five highest-activating training examples, retaining their three most frequent request verbs. Fixed lexical rules score non-execution, execution, discourse, and schema cues together with the activation-side/write-direction pattern. A candidate receives the highest-scoring family if its score reaches 4.0, or 3.0 for `schema_boundary_formatting`. This assigns 426 candidates; all 640 candidates and the remaining dictionary features retain their contributions in the full-feature K accounting.

Table 19: Formation-window feature families selected on training pairs. Side is the majority activation side on training pairs. Counts cover 426 of 640 candidates; 214 remain unlabelled. Mean and sum use held-out $|\kappa|$.

Family	Side	N_f	Mean	Sum	Interpretation
`analysis_non_execution`	corrupt	115	0.0633	7.2787	Analysis requests and non-necessity semantics.
`execution_request`	clean	292	0.0156	4.5634	Execution requests and action verbs.
`analysis_plain_discourse`	corrupt	15	0.0020	0.0294	General analysis discourse.
`schema_boundary_formatting`	corrupt	4	0.0466	0.1862	Code and format boundaries.

The family comparison asks which request-associated writes account for the analysis-side contribution mass. Although `analysis_non_execution` contains fewer labelled features than `execution_request`, its mean $|\kappa|$ is about four times larger, giving it the largest family total (Table 19). The concentration therefore comes from stronger contributions per feature. For example, L23 F117270 projects strongly onto tokens such as *nowhere*, *none*, *neither*, and *nothing*, and its highest-activating training examples use *Discuss* and *Explore*. Together with the small contribution of `analysis_plain_discourse`, these cues associate the dominant labelled writes with non-execution.

The decoder directions explain how these analysis-side contributions enlarge the paired gap. On the training pairs, 102 of the 115 `analysis_non_execution` features have higher analysis-side activation and a decoder projection onto $\hat{\mu}_\Delta$ that is negative. Greater activation therefore makes the analysis-state write more negative; subtracting that analysis state from the execution state gives a positive contribution to the gap. Combined with the high calling baseline under neutral requests (Appendix D.1), these signs support active suppression of a prepared call tendency on the analysis side. The feature contexts identify the requests recruiting this computation, and the signed writes explain its effect on the call state.

We next test whether the features identified by this signed decomposition exert behavioral influence under the same analysis requests. If their writes implement suppression, removing them should move the L24 state toward execution and increase calling. We zero the prediction-position writes of the five analysis-higher features with the largest training $|\kappa|$ whose decoder directions oppose $\hat{\mu}_\Delta$. These five features constitute less than 0.001% of the formation-window dictionary, yet their removal moves the state along $+\hat{\mu}_\Delta$ and raises calling from 3.0% to 25.5% (Table 20). The state and behavioral changes follow the predicted direction, connecting the decomposition to causal suppression; the layer-matched comparison ties the effect to the selected writes.

D.5 Linking Formation-Window MLP Writes to the Call State

The feature ablation identifies individual writes with a suppressive effect. We next ask whether the formation window's aggregate MLP effect on calling passes through the L24 call-direction coordinate. We replace all L20–L23 prediction-position MLP outputs with their paired opposite-side outputs, then restore only the original projection along $\hat{\mu}_\Delta$ at L24. The orthogonal component retains its

Table 20: Suppressor-write ablation on 200 feature-evaluation pairs, with 6 baseline analysis-side calls (3.0%). $\Delta_{\parallel}$ is the change in the L24 projection along $\hat{\mu}_{\Delta}$; gap closure is relative to the baseline execution–analysis gap. Call is the post-intervention call rate; strict recovery counts baseline text decisions changed to calls, divided by all 200 pairs.

Ablated writes	$\Delta_{\parallel}$	Gap closed (%)	Call (%)	Strict recovery (%)
Five training-ranked suppressors	+5.30	8.82	25.5	22.5
Layer-matched random suppressors	+0.14	0.23	2.5	0.5

swapped value, so the rescue tests whether restoring that single coordinate can recover the original call decision after a much broader perturbation.

On execution prompts, swapping the MLP outputs reduces calling to 93.0%. Restoring only the original call-direction projection recovers 100.0% calling, compared with 58.0% after orthogonal restoration (Table 21). The execution decision therefore recovers while the state's orthogonal component retains its swapped value. In the reverse arm, restoring the analysis-side projection lowers calling from 95.0% after the swap to 0.0%. The selective effects in both directions connect formation-window MLP computation to tool calling.

Table 21: Formation-window mediation on the 200 feature-evaluation pairs. Each arm computes g_l against the untouched paired opposite-side state. Call rate is the post-intervention top-1 rate.

	Execution prompts		Analysis prompts	
Condition	g_l	Call (%)	g_l	Call (%)
Baseline	60.06	100.0	60.06	3.0
MLP-output swap	15.78	93.0	15.09	95.0
Swap + $\hat{\mu}_{\Delta}$ restoration	60.06	100.0	60.06	0.0
Swap + orthogonal restoration	15.78	58.0	15.09	99.5
Swap + full-state restoration	60.06	100.0	60.06	3.0

E Reading Out the Tool-Call Vector

Downstream attention heads and MLPs translate the L24 call state into support for the first output token. We examine how components in L25–L35 respond to this state, use scaffold information, and contribute to call initiation.

E.1 Attention Heads and MLPs Contribute at Multiple Readout Stages

For each downstream attention head and MLP block, we compare direct output support with the effect of an intervention. The clean-minus-corrupt DLA delta measures the difference in the component's direct contribution to the `<tool_call>` logit between execution and analysis prompts. Single-component patching replaces an analysis-side output with its paired execution-side output and measures the resulting logit shift after downstream computation resumes. Strict flip is the fraction of all evaluated pairs whose baseline analysis response is text and whose patched response opens a call. Reverse patching measures the logit drop on execution prompts. Comparing these measurements relates each component's direct write to its influence through the remaining computation.

The contrast between DLA and patching reveals complementary roles in call readout. L25H6 and L28H3 have small DLA differences but substantial patching recovery, showing that their effects extend through the remaining computation (Table 22). The L33 heads show larger differences in direct output support. These results distinguish the influence of earlier components on downstream processing from the direct call-token support supplied by later writers.

The MLP results place effective readout contributions at several downstream layers. MLP34 gives the highest recovery in the component sweep, while MLP25 and MLP27 also produce large effects (Table 23). This distribution connects the L24 state to calling through both earlier and later MLP computation. We focus the feature analysis on MLP34 because its strong patching effect and late call-directed write provide a useful site for resolving how the state is expressed in the output format.

Table 22: Selected attention heads in the L25–L35 component sweep (baseline corrupt call rate 3.6%), also shown in Figure 5A. DLA delta is clean minus corrupt for the <tool_call> logit. Recovery shift and strict flip measure replacement of the corrupt-side head output with its output.

Head	stage	DLA Δ	recover shift	corrupt top-1	strict flip
L25H6	earlier	0.06	1.28	40.4%	36.9%
L28H3	earlier	0.26	1.38	36.1%	32.5%
L26H14	earlier	0.04	0.91	24.3%	20.7%
L29H9	earlier	4.15	1.34	21.3%	17.8%
L33H29	late	7.91	0.62	8.5%	4.9%
L33H11	late	3.32	-0.39	2.4%	0.0%

Table 23: Top MLP blocks in the L25–L35 component sweep (baseline corrupt call rate 3.6%). Recovery replaces a corrupt-side MLP output with its paired clean-side output; the reverse replacement on clean prompts gives the drop measurement.

MLP	recover shift	drop shift	corrupt top-1	strict flip
L34	1.74	2.32	53.3%	49.7%
L27	1.32	0.82	52.3%	48.7%
L25	1.70	0.96	44.8%	41.2%
L35	0.84	1.27	31.8%	28.2%
L31	0.90	0.90	28.6%	25.0%
L29	0.38	1.58	26.6%	23.1%

E.2 The Call State Regulates Downstream Call-Token Writes

L29H9, L33H11, and L33H29 attend more strongly to the tool-format template on execution prompts, although the scaffold is identical within each held-out pair (Figure 5B). We ask whether changing the internal call state can recruit these heads' call-directed writes while the request and scaffold remain fixed. Adding μ_Δ at L24 on analysis prompts brings all three heads' call-token DLA close to their execution-side values (Table 24). The same input therefore supports stronger writes toward the call opening when the intermediate state is shifted toward calling.

To test the readout in the suppression direction, we subtract μ_Δ at L24 on execution prompts in a separate attribution control. Total L30–L35 call-token DLA falls by 35.45%, with the largest absolute drops in L33–L35. The weaker output writes occur downstream of the intervention while the prompt remains fixed. Together with the head-level response to addition, this shows that changing the intermediate call state regulates how strongly subsequent components support the call-opening token.

Table 24: Attention-head response to L24 vector addition on the 200 balanced Qwen3-8B held-out pairs. Entries are mean DLA for the call-opening token.

Head	Clean	Corrupt	Corrupt $+\mu_\Delta$
L29H9	6.99	1.43	6.91
L33H11	5.60	1.46	5.30
L33H29	24.58	11.13	24.43

E.3 MLP34 Feature F109925 Supports the Tool-Call Opening

F109925 offers a concrete link between the residual state and the output format. Its max-activating contexts highlight tool-schema boundaries, and its decoder writes toward the call-opening token. Adding μ_Δ to analysis prompts restores feature activation toward its execution-side level, increasing call-directed output through that fixed decoder (Table 25). The state intervention therefore recruits a feature capable of supporting the call opening, while the activating contexts provide a structural interpretation of its write.

To test whether this structural write affects the decision, we replace F109925's analysis-side activation at the prediction position with its paired execution-side value through the feature decoder, using the native normalized MLP input. The replacement yields 37.0% calling, approaching the full MLP34

Table 25: L34 F109925 on the 200 Qwen3-8B feature-evaluation held-out pairs. Projected write is activation multiplied by the feature decoder's projection onto the call-token unembedding direction.

Measurement	Clean	Corrupt	Corrupt $+\mu_\Delta$
Mean activation	124.43	67.63	126.00
Projected call-token write	6.17	3.35	6.25

patch's 41.0%, with 6.5% calling for the fixed non-structural control F91365 (Table 26). Changing this single feature supplies enough call-directed support to cross the decision boundary on many prompts. Together with its activation response to μ_Δ, the behavioral effect identifies a structural write through which the call state promotes the call opening.

Table 26: L34 feature replacement and controls on the same 200 held-out pairs. All interventions act at the prediction position.

Condition	Call (%)
Corrupt $+\mu_\Delta$ at L24	100.0
Replace F109925	37.0
Replace F91365	6.5
Patch full MLP34	41.0

F Cross-Model Timing and Scaffold Attention

These diagnostics relate the model-specific call states in Table 6 to the timing of request encoding and the use of native scaffold information.

F.1 Early Request Encoding and Late Call-Token Writes Across Models

We measure request separability and call-token expression across all seven models to trace their progression toward the output decision. The held-out diagnostic cohorts contain 200 pairs for Qwen3-8B, 240 for Granite-3.3-8B, and 300 for each of the other five models (Table 27). For each model, we measure: (i) the linear probe AUC along the residual stream to detect when clean and corrupt representations become linearly separable; (ii) the logit-lens gap $\Delta\text{logit}(l) = \text{logit}_{\text{clean}}(l) - \text{logit}_{\text{corrupt}}(l)$ projected onto the model's native tool-call opening token unembedding direction; and (iii) the layer-wise Direct Logit Attribution (DLA) $\Delta\text{DLA}(l) = \text{DLA}_{\text{clean}}(l) - \text{DLA}_{\text{corrupt}}(l)$ of each block's net residual write, combining attention and MLP outputs, to measure direct support for the call token.

Table 27 summarizes the timing landmarks across all seven models, and Figure 6 displays the full layer-by-layer curves.

Table 27: Early-signal and timing diagnostics across all seven models on held-out prompt pairs. L^* is the causal intervention layer from Table 6; Probe AUC > 0.9 is the first layer where the linear probe achieves > 0.9 AUC; Gap > 1.0 is the first layer where the logit-lens clean-minus-corrupt gap exceeds 1.0; Top-3 DLA reports the blocks with the largest signed positive net direct logit attribution differences ($\Delta\text{DLA} > 0$), combining attention and MLP writes. L^* denotes a block input; probe and lens layer labels denote block-output residuals, and DLA labels denote block writes.

Model	L_{tot}	L^*	N	Probe AUC > 0.9	Gap > 1.0	Top-3 DLA layers
Qwen3-4B	36	L26	300	L0	L26	L34, L33, L32
Qwen3-8B	36	L24	200	L1	L27	L34, L33, L29
Qwen3-14B	40	L34	300	L1	L25	L34, L37, L38
Qwen3.5-4B	32	L31	300	L0	L14	L31, L30, L27
Qwen3.5-9B	32	L31	300	L0	L21	L27, L30, L23
Mistral-Small-3.2-24B	40	L25	300	L0	L17	L39, L37, L34
Granite-3.3-8B	40	L35	240	L1	L16	L39, L38, L37

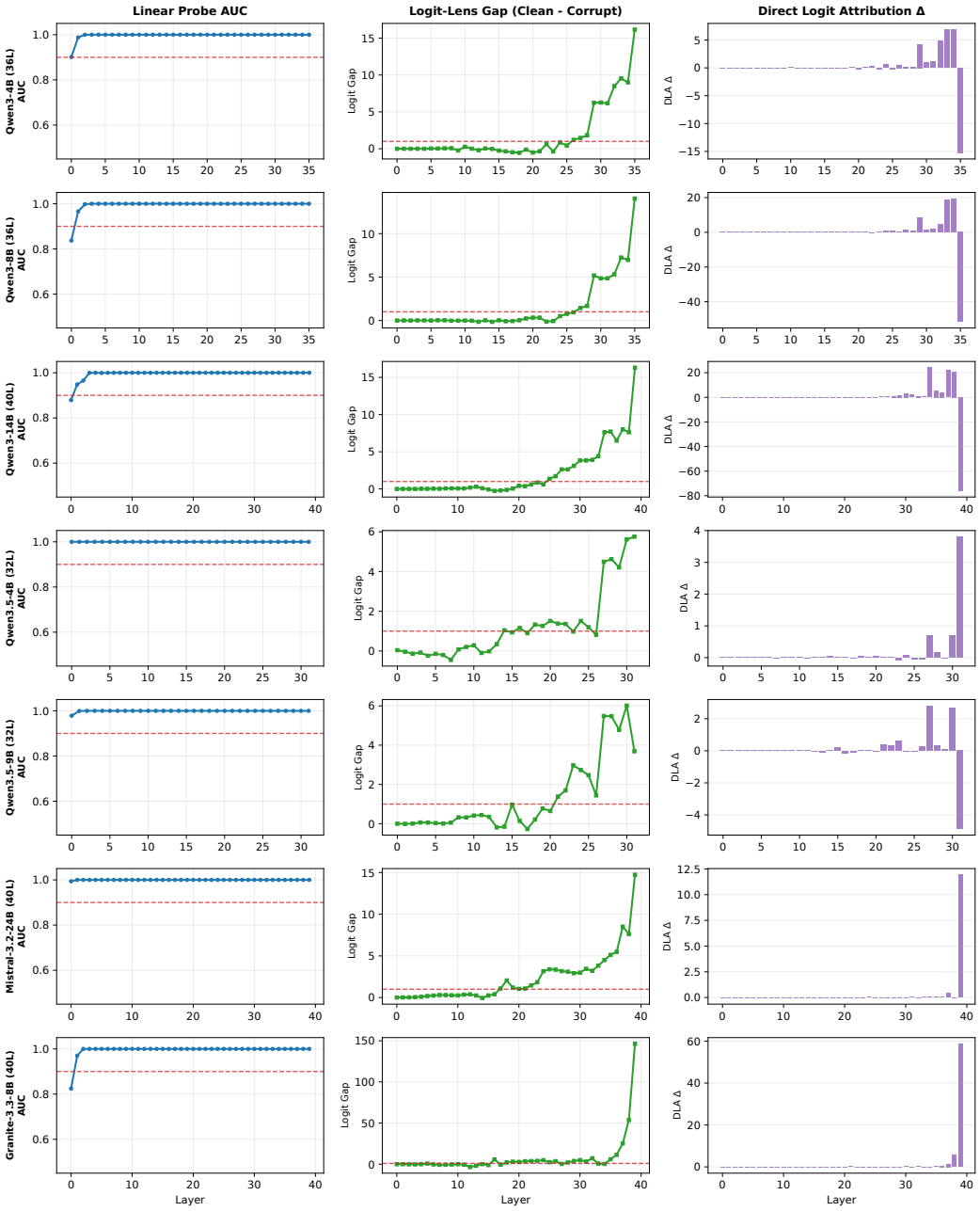


Figure 6: Triplet diagnostics across all seven models, arranged with one model per row and three diagnostics per row. From left to right, each row shows the linear probe AUC, the logit-lens clean-minus-corrupt gap, and the layer-wise DLA sweep. Evaluation cohorts contain 200 pairs for Qwen3-8B, 240 for Granite-3.3-8B, and 300 for each of the other five models. Across all seven models, probe separability appears at L0–L1, while the largest DLA writes occur in upper layers.

Across all seven models, request information is recoverable at L0–L1, while the strongest direct call-token writes occur in upper layers. This shared separation places request encoding early enough to guide substantial subsequent computation, with the largest direct expression of the contrast concentrated near the output. Together with the interventions in Table 6, these diagnostics connect early request information to a causally controllable call state and late output support across architectures.

The relation between vocabulary alignment and the effective intervention state varies across models. Qwen3-8B supports call control at L24 before its lens gap crosses 1.0 at L27; Granite crosses that threshold at L16 and supports intervention at L35 (Table 27). These measurements expose complementary aspects of the computation: the lens tracks developing output preference, and the intervention identifies a state through which the remaining network can be directed toward a different call decision. Their model-specific timing accompanies the early-encoding and late-writing pattern.

F.2 Execution Requests Increase Attention to Native Tool Scaffolds

Native scaffold attention tests how the request contrast changes the use of the tool interface already present in the prompt. Max Attn in Table 6 averages each measured head's execution-minus-analysis attention shift over native held-out pairs, then takes the largest head-level mean. The target region is the invocation-format template for Qwen-family models and the tool-schema region for Mistral and Granite. Qwen3-8B uses its three reported scaffold readers; Qwen3.5 measurements use full-attention layers.

The positive shifts connect execution requests to increased attention to call-relevant scaffold content. Because the scaffold stays fixed within a pair, the change reflects how the request recruits the available interface information. This extends the Qwen3-8B attention analysis to each model's native serialization: format instructions and tool definitions provide the scaffold information through which a call is expressed.

G Circuit Discovery Diagnostics

We use ACDC and EAP-IG to examine which component sets and interactions sustain the call contrast. Their replay and removal controls relate the compact residual-state intervention to the distributed computation supporting the decision.

G.1 ACDC: Sparse-Core Replay and Removal Across Qwen3 Scales

ACDC [9] prunes candidate component connections according to their effect under activation patching. We evaluate the resulting sparse forward cores across Qwen3 scales, including the additional Qwen3-1.7B diagnostic.

We compare a core's ability to carry the task signal under replay with its contribution in the intact network (Table 28). Sufficiency patches the selected clean component outputs into the analysis run; removal replaces those outputs in the clean run with their paired analysis-side values. Let s_{clean}, s_{analysis}, s_{replay}, and s_{removal} denote the corresponding next-token task scores. For each pair, sufficiency is $(s_{\text{replay}} - s_{\text{analysis}})/(s_{\text{clean}} - s_{\text{analysis}})$, and the removal effect is $(s_{\text{clean}} - s_{\text{removal}})/(s_{\text{clean}} - s_{\text{analysis}})$. We report mean ratios and compare replay with size-matched random component sets.

Table 28: Sparse forward-core replay across Qwen3 scales. Core nodes and edges are taken from the aggregated forward sparse graph. Suff. is the mean normalized score recovery under core replay; Removal is the mean normalized score reduction under clean-side replacement of the core. Rand. Suff. is the mean recovery for size-matched random component sets, and Gap is the discovered core's mean advantage over that baseline.

Model	Core nodes	Core edges	Suff.	Removal	Rand. Suff.	Gap
Qwen3-1.7B	18	54	0.9891	0.9824	0.8812	0.1078
Qwen3-4B	18	69	0.9979	0.8319	0.8712	0.1266
Qwen3-8B	18	63	1.0000	0.5975	0.9768	0.0232
Qwen3-14B	20	86	0.9999	0.4077	0.9554	0.0446

The cores recover scores close to the clean baseline under replay at every tested scale. At 8B and 14B, the size-matched random sets also recover much of that score, so the replayed signal can pass through several component sets. Clean-side replacement reveals the selected cores' influence within the full computation: the normalized score reductions are 0.5975 and 0.4077, respectively (Table 28). The combination links effective sparse replay to contributions spread across the intact network, motivating the shared residual state as a way to trace their collective effect on calling.

G.2 EAP-IG: Recovery of the Call Contrast Across Edge Budgets

To examine how recovery changes as more of the computation is retained, we rank edges with EAP-IG [26]. Integrated gradients estimate the effect of moving edge activations between paired analysis and execution runs; retaining top-ranked edges or greedily connected edges gives circuits at different budgets. Gradient-aware variants further refine this attribution procedure [27]. We evaluate each retained graph by its recovery of the call-related logit-difference score, normalized between the analysis and execution baselines.

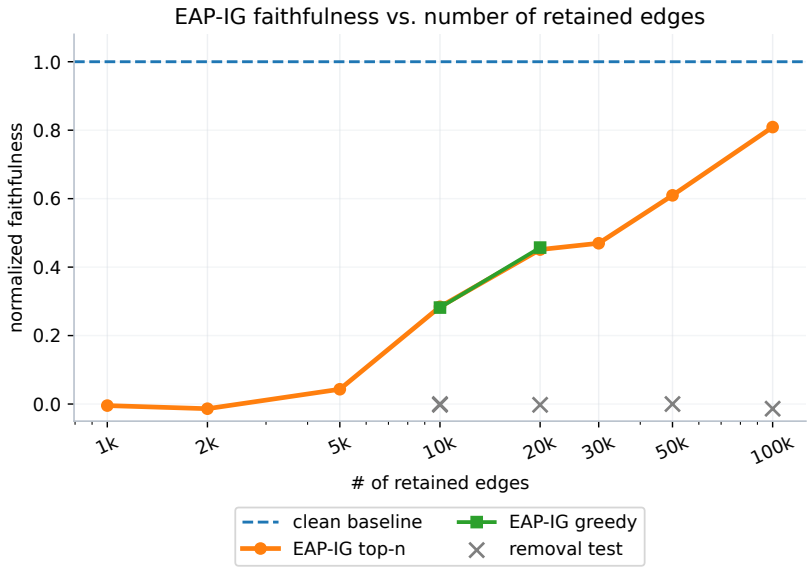


Figure 7: EAP-IG top-k faithfulness curve for the tool-call task, normalized between the corrupt and clean baselines. The curve relates retained edge budget to recovery of the `<tool_call>` signal.

Table 29: EAP-IG top-k sweep on Qwen3-8B using 100 equal-length clean/corrupt prompt pairs. Faith. normalizes sufficiency between the corrupt and clean baselines. Suff. is the mean logit-difference score under corrupt-side replay with only the retained edges active. Nodes counts the retained component set induced by the edges; core overlap counts shared nodes with the 18-node forward core.

Retained edges	Faith.	Suff. logit diff	Nodes	Core overlap
1,000	-0.005	-13.12	139	17/18
2,000	-0.014	-13.38	229	18/18
5,000	0.043	-11.81	417	18/18
10,000	0.284	-5.19	553	18/18
20,000	0.451	-0.59	708	18/18
30,000	0.470	-0.08	805	18/18
50,000	0.610	3.77	937	18/18
100,000	0.809	9.25	1,075	18/18

The full 18-node forward core is already represented at 2,000 retained edges, before substantial task recovery appears. As the retained computation expands in both nodes and edges, faithfulness rises to 0.809 at 100,000 edges (Table 29). Thus the contribution of the core develops through a broader

set of components and interactions. Clean-side removal of the retained edges also strongly reduces the task score (Figure 7), linking that expanded computation to the call contrast. These replay and removal results complement the vector intervention by locating distributed support for the shared call-controlling state.

H Related Work

This section relates the tool-call decision to three lines of work: agentic tool-use systems, mechanistic analysis at the representation, circuit, and feature levels, and behavior-mediating residual directions.

H.1 Tool-Calling and Agentic Systems in Large Language Models

Training and prompting for tool use. Enabling large language models to interact with external environments has progressed across several training and prompting paradigms. Early methods introduced tool-calling capabilities by fine-tuning models on synthetically generated or bootstrap-filtered tool execution trajectories [1, 4, 5, 28]. Concurrent prompting frameworks established multi-step interaction protocols, most notably ReAct [2], where models alternate between reasoning traces and environment actions. Modern open-weight foundation models, such as the Qwen family [3], IBM Granite, and Mistral-Small, incorporate dedicated agentic pre-training and instruction tuning, allowing models to natively output structured special tokens (e.g., `<tool_call>`) and parse tool response blocks without external wrapper scripts.

Evaluation benchmarks and execution environments. Evaluation suites span single-turn API selection, stateful tool use, and agentic instruction following. MetaTool evaluates tool awareness and selection [6], while BFCL covers function calling and stateful multi-step agentic evaluation [7]. τ-Bench [29] and τ^2-Bench [19] evaluate sequential interaction in simulated domains, while AgentIF focuses on instruction following under long and complex agentic constraints [11]. Code-generation benchmarks, including MBPP [12], APPS [13], HumanEval [14], and CodeContests [15], provide naturally structured environments where a tool interface (e.g., `write_file` or an interpreter) is embedded directly into complex problem specifications. Robustness evaluations further reveal that tool-calling models can be sensitive to superficial prompt variations, format drift, and adversarial perturbations [30–32].

Behavioral outcomes versus internal mechanisms. Tool-use benchmarks measure tool selection, argument construction, action accuracy, and task completion. Our analysis examines the internal computation leading to the first-token choice that initiates a tool call, connecting a specific stage of these broader behaviors to a causally controllable residual state.

H.2 Mechanistic Interpretability: Representations, Circuits, and Features

Mechanistic interpretability aims to reverse-engineer neural networks into human-understandable computational graphs and representational states. We contextualize our methodology across three distinct levels of granularity: the representation level, the circuit level, and the feature level.

Representation-level localization and causal interventions. Representation-level methods connect internal states to behavior through complementary measurements. Linear probes test recoverability across layers [33–35], while vocabulary projections such as the logit lens and tuned lens track developing token preferences [36]. DLA resolves direct output-logit contributions [21], and activation patching and interchange interventions test the effects of replacing internal states [17, 18]. Their combination supports the distinction between request information, causal call control, and downstream output support used in our analysis.

Circuit-level analysis of tool-call computation. Circuit analysis identifies component interactions that reproduce a task computation [8, 9, 37]. Automated methods such as ACDC and Edge Attribution Patching map candidate paths through attention heads and MLPs; gradient-aware discovery [27] and self-influence analysis [38] further examine how these interactions support behavior. Tool-call initiation integrates information from request wording, task content, tool definitions, and invocation

templates. Circuit diagnostics describe how this information contributes across components (Appendix G), while the residual-state intervention tests their shared decision-level effect. Transcoder decomposition then assigns signed contributions to the features forming the state, connecting component interactions to a causally controllable representation.

Feature-level decomposition via Transcoders. Sparse dictionary methods resolve MLP computation into feature activations and writes [39]. SAEs reconstruct internal activations [40–43], and Transcoders model the MLP input-to-output mapping as sparse activations multiplied by decoder directions [20]. This factorization connects the contexts recruiting a feature to the direction in which it changes the residual state. It supplies the signed write analysis linking request-sensitive formation features and structural readout features to the call state. Work on planning features across scales [44] further motivates comparing this organization across model families.

H.3 Behavior-Mediating Directions and Steering Vectors

Representation engineering and activation steering. A growing body of literature demonstrates that complex model behaviors can be mediated and controlled by compact directions in residual-stream space. Representation engineering [22] extracts behavioral concepts via contrastive pairs and injects them to control model attributes. Related steering techniques, including activation addition [45], Contrastive Activation Addition (CAA) [46], and Inference-Time Intervention (ITI) [47], show that targeted vector additions can alter truthfulness, sycophancy, or stylistic persona without retraining model weights. Recent frameworks further refine steering precision and stability: Jiang et al. [48] introduce Global Evolutionary Steering to enforce cross-layer consistency, Yu et al. [49] develop PIXEL for adaptive position-wise injection with subspace calibration, and Yu et al. [50] propose MARI to calibrate representation interventions across multi-adapter architectures. Similarly, function vectors [23] demonstrate that in-context task execution routines compress into localized vectors that can trigger specific behaviors when injected into zero-shot contexts.

The refusal direction and default-and-suppression dynamics. The closest conceptual precedent to our findings is the refusal direction described by Arditi et al. [16]. They show that refusal in safety-aligned language models is mediated by a single residual-stream direction: ablating this direction suppresses refusal, while adding it can induce refusal on harmless requests. Our analysis reveals an analogous functional organization in tool calling: the prompt scaffold establishes a strong call prior on task-bearing inputs, while analysis requests activate suppressor features that write against $\hat{\mu}_{\Delta}$ to inhibit call initiation (Section 5).

The transfer experiments connect this mechanistic account to the reuse of steering directions. A direction fitted in the controlled request contrast remains effective under new tasks, interfaces, and request forms: cross-domain tests calibrate its magnitude on separate target-domain training pairs, while the τ^2-Bench and verb-free tests use fixed gains of the original vector. This connects the state-level intervention to a reusable action-initiation signal.

I Implementation and Reproducibility

The implementation uses PyTorch and Hugging Face Transformers. Frozen prompt manifests record the construction models, data splits, and file hashes. Model-specific templates preserve the native tool-call serialization.

I.1 Evaluation Protocol

Vector fitting uses training pairs only. Evaluation applies interventions at the final input-token position and measures the logits of the first output token. Layer indices are zero-based, and L^* denotes the block-input residual. The localization sweep compares the request-verb and prediction positions across decoder layers. The Qwen3-8B attention and feature-response analyses cover L25–L35 on the 200 balanced held-out pairs. Qwen3.5-4B formation-feature mass is evaluated over L28–L30. Cross-model attention measurements use each model's native held-out inputs and native call-opening token. The main paired experiments use a fixed 300/200 training/held-out split and the fixed block-input intervention layers. Additional appendix controls use the cohorts specified in their subsections.

Cross-domain transfer keeps the fitted direction fixed and calibrates its magnitude on target-domain training pairs. For Qwen3, Qwen3.5, Mistral, and Granite, intervention rates are evaluated on separate held-out pairs. The τ^2-Bench and verb-free tests apply fixed gains to the original fitted vector.

I.2 Code, Data, and Intermediate Artifacts

The release comprises experiment code, the model-specific paired datasets, and intermediate artifacts needed to reproduce the conclusions, including Transcoder checkpoints. The repositories are:

- Code and experiment instructions: https://github.com/XijieGo/MI4ToolCalling.
- Paired datasets and manifests: https://huggingface.co/datasets/XijieGong/MI4ToolCalling.
- Newly trained Transcoders and checkpoint documentation: https://huggingface.co/XijieGong/MI4ToolCalling.

Repository README files and per-model manifests document the directory layout and input formats; the Transcoder model card provides checkpoint descriptions and loading instructions. Appendix A documents the prompt construction and per-model splits.

Our original code, dataset-construction materials, and newly trained Transcoders use Apache License 2.0. Source benchmark content and third-party checkpoints retain their original licenses, summarized in Table 30. The base models in the primary comparison are Qwen3-4B, Qwen3-8B, Qwen3-14B, Qwen3.5-4B, Qwen3.5-9B, Mistral-Small-3.2-24B-Instruct-2506, and Granite-3.3-8B-Instruct. All seven use Apache License 2.0. The Qwen3 Transcoders of Hanna and Ameisen [44] are available in the `mwhanna/qwen3-{4b,8b,14b}-transcoders` repositories under the MIT license. PyTorch uses the BSD 3-Clause license, and Hugging Face Transformers uses Apache License 2.0.

Table 30: Licenses of the source benchmarks. Links identify the original resources; the original papers are cited in the dataset and transfer sections.

Source benchmark	License
APPS	MIT
HumanEval	MIT
MBPP	CC BY 4.0
CodeContests	CC BY 4.0 (data); Apache 2.0 (code)
τ^2-Bench	MIT
FEVER	CC BY-SA 3.0 (data)
Spider 1.0	CC BY-SA 4.0 (data); Apache 2.0 (code)

I.3 Compute Resources

Training our Transcoders required approximately 800 NVIDIA B200 GPU-hours in total, including debugging and production training. Using the provided checkpoints, the other reported experiments can be reproduced on a single NVIDIA RTX PRO 6000 Blackwell GPU with 96 GB of memory.

I.4 LLM Usage

GPT-5 was used to generate debugging examples for the data-construction pipeline. The models evaluated in the mechanistic experiments are documented in Appendix I.2; the additional Qwen3-1.7B replay diagnostic is specified in Appendix G.1.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main claims: a controlled first-token decision interface, a tool-call vector that transfers across settings, and an account of the vector's formation and downstream readout. Scope and boundary conditions are further clarified in Limitations.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated Limitations section that defines the scope as the binary call-or-no-call decision and identifies the unresolved fine-grained formation mechanism.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper is empirical and does not present formal theorems or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The main text defines activation patching, the shared vector μ_Δ, and the sufficiency and necessity metrics. Appendix A describes the paired datasets, prompt formats, and per-model splits. Appendix I identifies the experiment code, datasets, and Transcoder checkpoints needed for reproduction. And we release the them.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The reproducibility package contains experiment code, paired datasets, and the intermediate artifacts needed for reproduction, including Transcoder checkpoints. Appendix I.2 provides the GitHub and Hugging Face repository links and identifies the accompanying README files, input manifests, and checkpoint loading instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main paper and Appendix A describe the model-specific prompts, source families, and data splits. Intervention locations and evaluation set sizes accompany the corresponding results. Appendix I and the repository documentation describe the evaluation protocol and checkpoint access.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The main causal results are point estimates on fixed held-out sets. We report evaluation sample sizes and norm-matched random controls, but do not report confidence intervals or significance tests for these results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix I.3 reports approximately 800 NVIDIA B200 GPU-hours for Transcoder training, including debugging and production training. Using the provided checkpoints, the remaining experiments can be reproduced on a single NVIDIA RTX PRO 6000 Blackwell GPU with 96 GB of memory.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. It uses offline benchmark-based evaluation and causal model interventions, without human-subject studies or collection of sensitive personal data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section 10 discusses potential benefits for understanding and regulating tool use, as well as the possibility of inducing calls inconsistent with user intent.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The new assets are benchmark-derived paired prompt datasets, analysis code, and Transcoders for interpreting existing models. These research assets do not pose the high-risk release scenarios described in this question.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

Justification: Original models, benchmarks, and Transcoders are credited in the paper and Appendix I.2, which lists their licenses and source links. Our original release materials use Apache License 2.0; source benchmark content and third-party checkpoints retain their original licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces paired datasets with 500 clean/corrupt pairs for each of the seven primary models. Appendix A documents their sources, splits, prompt formats, and construction. README files, per-model manifests, and the Transcoder model card accompany the assets in the repositories linked in Appendix I.2.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects or participant studies.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Appendix I.4 discloses the use of GPT-5 to generate debugging examples for the data pipeline. The primary mechanistic experiments study seven models from the Qwen3, Qwen3.5, Mistral, and Granite families (Appendix F); the additional sparse forward-core replay diagnostic also includes Qwen3-1.7B (Appendix G.1).

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.