Tool Calling is Linearly Readable and Steerable in Language Models

Anonymous ACL submission

Abstract

When a tool-calling agent picks the wrong tool, the failure is invisible until execution: the email gets sent, the meeting gets missed. As agents take on consequential actions (running code, moving money, approving transactions), one bad tool call can do real damage, and we currently have no way to look inside the model and catch the mistake before it happens. This paper shows that we can. Inside the model, the choice of tool is carried by a single direction in activation space, one direction per pair of tools. Adding that direction during generation switches which tool the model picks. Across 12 instruction-tuned and 6 base models spanning Gemma 3, Qwen 3, Qwen 2.5, and Llama 3.1 (270M to 27B), this works at 83–100% accuracy on 4B+ instruction-tuned models on a 15-tool synthetic benchmark and at 77–94% on the real-API benchmark τ-bench airline. The JSON arguments that follow automatically adapt to the new tool's schema, so flipping the name is enough. The same per-tool directions also flag likely errors before they happen: queries where the model's top two tool candidates are nearly tied fail 21$\times$ more often than queries with a clear top choice (Gemma 3 27B). This is not just steering toward a different topic: random vectors at the same magnitude give a 0% switch rate, a probe within a single domain (14 airline tools that all share one topic) still reads which tool the model will call at top-1 61–89% across five 4B–14B models, and the causal effect concentrates along the model's own first-token output direction. Even base models already carry the right tool internally before they can emit it: reading the chosen tool off the model's internal state (cosine readout) recovers 61–82% accuracy on the BFCL function-calling benchmark while base generation lands at 2–10%, suggesting pretraining forms the representation and instruction tuning later wires it to the output. Our results cover single-turn, fixed-menu settings; on multi-turn agent loops the same intervention is less stable. The linear signal we identify gives both a practical hook for catching wrong tool calls before they execute and a mechanistic foothold for understanding how language models decide what to do.

1 Introduction

Imagine an LLM assistant is asked to "follow up with the client about tomorrow's meeting." It has access to `send_email`, `schedule_meeting`, and `search_contacts`. The model picks `send_email`, writes a plausible message, and fires it off. But the user wanted to *reschedule*, not send a reminder. The email goes out; the meeting is missed. This kind of silent failure is common: on the τ-bench airline benchmark (Yao et al., 2024), even 4B-parameter models only succeed 25% of the time, and most failures come down to picking the wrong tool (Schick et al., 2023; Qin et al., 2024). As agents get access to actions that matter (running code, moving money, approving transactions), one bad tool call can do real damage (Hendrycks et al., 2023).

The tricky part is that we currently have no idea *why* a model picks one tool over another. It outputs a tool name and some JSON arguments, but there is no way to peek inside and catch a mistake before it happens. Recent work has started looking at this: Healy et al. (2026) detect tool-calling hallucinations from hidden states, and Wang et al. (2026) improve binary decisions about whether to call a tool at all (F1: 0.18$\rightarrow$0.50). But neither of these traces the actual circuit responsible for selection, and neither shows how to control *which specific tool* gets chosen when there are many candidates. The question we ask is simple: **how do language models internally pick among tools, and can we intervene on that process?**

We investigate this by combining five interpretability methods across three model families

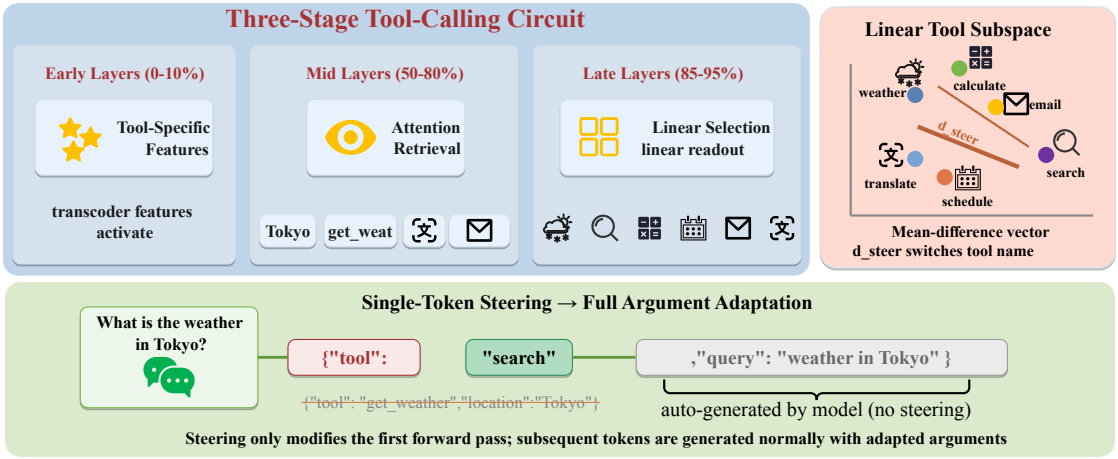

Figure 1: Overview of the three-stage circuit and steering demonstration. Adding a mean-difference vector redirects tool selection and automatically restructures arguments. Validated across 12 IT models in 3 families (Gemma 3, Qwen 3 / Qwen 2.5, Llama 3.1; 270M–27B).

(Gemma 3 (Gemma Team, 2024), Qwen 3 / Qwen 2.5 (Qwen Team, 2025), and Llama 3.1 (Grattafiori et al., 2024)), spanning 270M to 27B parameters across 12 instruction-tuned and 6 base models. Figure 1 gives the overview.

The core finding is simple: the choice of tool is carried by a single direction in the model's internal state, with one direction per pair of tools. Adding that direction switches which tool the model picks. Across 12 instruction-tuned and 6 base models spanning Gemma 3, Qwen 3, Qwen 2.5, and Llama 3.1 (270M to 27B), this works at 83–100% on 4B+ instruction-tuned models on a synthetic 15-tool benchmark using only tool names. Adding short tool descriptions makes at most a small difference on 4B+ models and hurts some sub-1B models, where the longer prompt overwhelms the smaller model (Appendix A.24). The JSON arguments that follow automatically adapt to the new tool's schema (Section 4.4). The same per-tool directions also flag likely errors before they happen: when the model is unsure between two tools, it is much more likely to pick the wrong one (Section 5; correction in Appendix A.9).

We are careful about the geometric framing. PCA over 15 tool means fits in about 10 directions, but a matched-prompt control where we keep the 15-tool prefix and only change the user query gives the same compactness, so the 15-tool dimensionality on its own is not evidence of tool-specific compression (Section 4.1). At 200 real APIs from ToolBench the picture is similar: 36 directions cover the tools but the matched control sits at 39, so part of what looks like tool-specific structure reflects the shared prompt prefix. The interventional results, not the dimensionality, are what carries the linearity claim.

Base (pretrained-only) models extend this picture. On BFCL v3, a Gemma 3 4B base model generates the correct tool on only about 3% of queries, but reading the closest mean-activation direction from its residual stream recovers 75%; the gap is similar on Gemma 1B, 12B, and Llama 3.1 8B base models (56–72 percentage points). Instruction tuning, it seems, mostly wires the existing internal tool signal into the output layer, rather than creating the signal.

To understand *how* this works, we trace the components that push the model toward the right tool (Section 4.2). Patching components in and out one at a time, we find a rough three-stage pathway: dedicated features light up at early layers when a specific tool is relevant, attention heads at mid-layers pull in the right context, and late layers make the final pick. A few specific attention heads carry the decision on Gemma (like L17 H0 and H1 in Gemma 3 4B), but the picture is model-dependent: on Qwen 3 4B the top-k heads and top-k MLPs are essentially tied (Section 4.2, Table 22). The circuit is not always present: it is absent at 270M parameters, starts emerging at 1B, and gets sharper with instruction tuning (Section 4.5).

2 Related work

Most work on tool use in LLMs focuses on making models better at calling tools, whether through specialized training (Schick et al., 2023; Patil et al., 2023; Qin et al., 2024), benchmarks (Yao et al., 2024; Li et al., 2023b), or prompt design (Hao et al., 2023). We are asking a different question: not how to improve tool use, but how it works internally. Two recent papers look in this direction. Healy et al. (2026) show that tool-calling hallucinations leave a detectable trace in hidden states, and Wang et al. (2026) steer binary tool/no-tool decisions (F1: 0.18→0.50). We go further, from binary to multi-class: steering among 15+ tools with structured output adaptation.

Our work builds on the idea that language models represent high-level concepts as directions in activation space, sometimes called the linear representation hypothesis (Park et al., 2024; Nanda et al., 2023). This has been confirmed for things like sentiment (Tigges et al., 2023), truthfulness (Marks and Tegmark, 2024), and refusal (Arditi et al., 2024), though not everything is linear (Engels et al., 2024). Several methods exploit this linearity to steer model behavior at inference time (Li et al., 2023a; Turner et al., 2024; Zou et al., 2023). The closest prior work to ours is by Todd et al. (2024), who find that in-context learning tasks are encoded as "function vectors." The key difference is that their vectors capture *task type* (like translation or capitalization), while ours capture *which specific tool to call from a fixed menu*, and our steering produces structured JSON output, not just a label change.

On the mechanistic side, we combine three tools to trace how the model makes its decision. Activation patching (Vig et al., 2020; Wang et al., 2023; Conmy et al., 2023) tests which components matter. Sparse autoencoders (Cunningham et al., 2023; Bricken et al., 2023; Templeton et al., 2024) identify individual features. Cross-layer transcoders (Ameisen et al., 2025) decompose each layer's activations into interpretable per-layer features. Geva et al. (2023) found that factual recall follows an "attribute then retrieve" pattern, and we see a loosely similar shape for tool selection. The superposition framework (Elhage et al., 2022) also predicts that discrete categories sit in nearly orthogonal directions, which fits with our ∼10-dimensional subspace for 15 tools.

3 Method: Mean-Difference Steering for Tool Selection

The idea is straightforward. If different tools produce different average activation patterns, then the difference between two tool averages gives us a direction in activation space that points from one tool to another. Adding that direction during generation should switch the model's tool choice. The method requires no fine-tuning or gradients, just a few example queries per tool and one forward pass to collect activations.

We test 12 instruction-tuned models spanning 270M to 27B parameters across three families: Gemma 3 (Gemma Team, 2024) at five scales (270M to 27B), Qwen 3 (Qwen Team, 2025) at five scales (0.6B to 14B) plus Qwen 2.5 (7B), and Llama 3.1 (Grattafiori et al., 2024) (8B), with base (pre-trained) variants for Gemma and Llama where available (Appendix A.22). Circuit tracing with transcoders uses Gemma 3 4B (34 layers, $d_{\text{model}} = 2560$) as the reference. Each prompt lists K tool definitions followed by a user query. Mechanism experiments (steering, circuit tracing, PCA at $K{=}15$) use controlled synthetic tools (Appendix A.2) where we can isolate individual variables like tool naming and query semantics. Scaling and validation experiments use real APIs from τ-bench (Yao et al., 2024) (14–16 tools), ToolBench (Qin et al., 2024) (up to 2,000 APIs from 49 domains), and BFCL v3 (Yan et al., 2024) (1,053 real-world queries). We also apply SAEs, activation patching, and PCA to understand the internal representations, but the steering itself needs none of that machinery. The method has three steps. First, for each tool t_i, we collect $n = 2$–3 queries that should trigger that tool and record the model's internal state (its "activation") $\mathbf{h}_i^{(j)} \in \mathbb{R}^d$ at the final token position in the second-to-last layer ℓ ($\ell = 33$ for Gemma 3 4B, $\ell = 35$ for Qwen 3 4B). The mean activation per tool is $\bar{\mathbf{h}}_i = \frac{1}{n}\sum_j \mathbf{h}_i^{(j)}$. Second, the steering direction from tool t_a to tool t_b is just the normalized mean difference:

$$\mathbf{d}_{a\to b} = \frac{\bar{\mathbf{h}}_b - \bar{\mathbf{h}}_a}{\|\bar{\mathbf{h}}_b - \bar{\mathbf{h}}_a\|} \tag{1}$$

We scale it by $\alpha \cdot \text{sep}_{a\to b}$, where $\text{sep}_{a\to b} = (\bar{\mathbf{h}}_b - \bar{\mathbf{h}}_a) \cdot \mathbf{d}_{a\to b}$ is the projection gap between the two tool means. We pick α from $\{0.5, 0.7, 1.0\}$ using a 5-pair validation set disjoint from the test pairs. Third, during generation we add this vector to the

residual stream at layer ℓ:

$$\mathbf{h}'_\ell = \mathbf{h}_\ell + \alpha \cdot \text{sep}_{a\to b} \cdot \mathbf{d}_{a\to b} \quad (2)$$

We only intervene at the final token position on the first forward pass. No weights change; subsequent tokens are generated normally. This extends activation addition (Turner et al., 2024; Arditi et al., 2024) from binary features to multi-class tool selection.

We count a steer as successful if the model's next-token prediction matches the target tool's name prefix. Because four tools share the prefix `get_` (weather, stock_price, news, directions), a single-token check cannot distinguish them; we report both prefix-match and exact-match (via 5-token generation) rates and note the difference explicitly. All accuracy claims are tested against random chance ($1/K$), with 95% confidence intervals. We use standard statistical corrections (Bonferroni for multiple comparisons, Cohen's h for effect sizes) to make sure the results are not flukes; full CIs in Appendix A.21.

4 Experiments

We organize our experiments in two parts. First, we look inside the model: what the internal geometry looks like (Section 4.1) and which components are responsible (Section 4.2). Then we use this understanding to intervene: switching the model's tool choice (Section 4.3), observing how the output adapts (Section 4.4), and tracing when the structure develops across model scales (Section 4.5).

4.1 Tool identity fits in a small space

For each tool we compute a mean activation vector at the last prompt token (the position about to emit the tool name), giving 15 vectors in $\mathbb{R}^{2560}$ for the 15-tool set. We then ask how many independent directions these vectors actually span using principal component analysis (PCA) over the 15 centred means (method in Appendix A.7; Figure 2).

Across all models the first 10 components capture about 91% of the variance, and k_{90} (the smallest number of components reaching 90%) is consistently 9–11 across families and scales, well below the theoretical max of 14 for 15 points. We read k_{90} as a rough measure of how many independent directions the model uses for tools rather than a precise feature count, since networks can pack overlapping concepts via superposition (Elhage et al., 2022).

A random-Gaussian baseline gives only 74% top-10 variance (z=96.4 vs. our 93%), but a matched-prompt K=15 control with non-tool topic queries gives k_{90}=7–9 across four models, slightly tighter than tools. So the K=15 spectrum on its own does not isolate tool-specific structure (Appendix A.27), and the cleaner evidence comes from the interventional results in later sections. Base models show comparable spectra (89–93% top-10), suggesting the geometry is already in place after pretraining.

The subspace also survives controls for the tool-name token and for list position. Replacing the 15 tool names with anonymous IDs leaves k_{90} at 9–11 (Appendix A.28), and shuffling the tool-list ordering across five permutations leaves each tool's mean direction at cos $\geq$0.96 (Appendix A.29). The description-topic reading is handled in §5.

A stronger test: does the compression hold as we add more tools? We scale up using real API definitions from ToolBench (Qin et al., 2024), sampling evenly across 49 domains (finance, health, sports, etc.) from a pool of 12,000+ APIs (Table 1).

K TOOLS	k_{90}	MAX	RANDOM	COMPRESS
50	17	49	43.0	35%
100	26	99	86.0	26%
200	36	199	167.0	18%
500	57	499	392.0	11%

Table 1: PCA scaling on Gemma 3 4B with real ToolBench APIs (stratified across 49 domains, with descriptions). k_{90}: dimensions needed to explain 90% of variance. Compression improves with scale.

The trend continues from our 15-tool result (k_{90}=10, Section 4.1): more tools require more dimensions, but the growth is much slower than random. At K=200, the model needs only 36 directions (18% of the theoretical max; random points would need 167).

Matched-prompt control. The random-Gaussian baseline is weak: any 200 means sharing a long prompt prefix will compress better than iid noise. A matched-prompt control swaps the user query for a non-tool topic with everything else held fixed (Table 26, Appendix A.27). The model-by-model picture is mixed (Gemma 3 4B indistinguishable from baseline, Qwen 3 4B and Gemma 3 12B with tools spreading wider, Llama 3.1 8B the reverse), so the K=200 spectrum on its own does not isolate tool-specific structure. Cleaner evidence comes from the matched K=15 control (Appendix A.27), the within-topic τ-bench airline

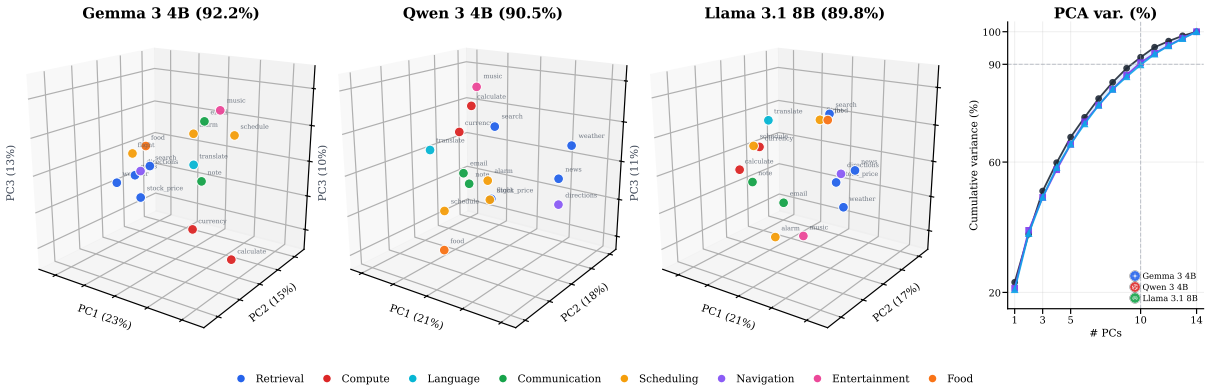

Figure 2: Left: 3D PCA of 15-tool activations, one model per family. Right: cumulative variance converges to ∼90% at 10 PCs. Cross-source (39 tools) in Appendix A.19.

probe (§5, top-1 61–89%), and the steering and patching results.

Prompt-length effects. Scaling is not perfectly smooth: at some K, k_{90} dips before rising, because real APIs cluster by domain (full curve in Figure 7, Appendix). Tool descriptions help compression at small K but lengthen the prompt enough to hurt at K>750 (Appendix A.24); they also raise cross-lingual classification from 60% to 93% (Appendix A.23).

4.2 Tracing the decision from input to output

We now know that tool identity fits in a small space, but we have not yet asked *how* the model builds that representation, step by step from input to output. We use the term "circuit" loosely to mean the localized pathway responsible for tool selection, not a fully specified computational graph in the sense of Conmy et al. (2023); we identify which components matter and how they contribute, but do not trace every edge.

We decompose layer activations with cross-layer transcoders (Ameisen et al., 2025; Lieberum et al., 2024), which split each layer into individual "features" (detectors for concepts like "the user wants weather" or "output a JSON brace"). On Gemma 3 4B a three-stage pipeline emerges: 38 tool-selective features fire in early layers L0–3, mid-layer attention heads in L16–30 lock onto tool-name and entity tokens (L24 head 1 attends to "Tokyo" at weight 0.43), and late-layer features at L30–33 handle JSON formatting through generic code-generation features shared with non-tool contexts (NeuronPedia cross-reference in Appendix A.5). Qwen 3 4B replicates the shape, with steering becoming effective at L23+ (per-layer details in Appendix A.12). The PCA subspaces are comparable across families (Gemma 92.5% vs. Qwen 90.2% in 10 PCs; Table 7, Appendix A.11).

Does this circuit actually matter, or is it just correlation? We test causality with activation patching (Conmy et al., 2023): for each attention head and MLP layer we swap in the output from a different-tool query and measure how far the model's confidence in the correct tool drops, averaged over 10 tool pairs (details in Appendix A.25).

The causal results line up with the three-stage structure. On Gemma 3 4B, two attention heads in the middle (L17 H0 and H1) matter more than every other head combined: swapping them drops confidence in the correct tool by 6.5 and 3.7 points. The three stages contribute 17%/37%/45% of the total effect on Gemma and 24%/33%/43% on Qwen. Summed across all components, attention totals 80–88% and MLPs 12–20%; this raw share is partly a counting artefact (Gemma 3 4B has 272 heads vs. 34 MLPs; Qwen 3 4B has 1,152 vs. 36). A fairer top-3 comparison (Table 22, Appendix A.25) gives heads 2.7× MLPs on Gemma 4B (a few specific heads dominate) but heads ≈ MLPs on Qwen 4B; the honest reading depends on the model. Either way, the localisation differs from prior factual-recall work, which is associated mostly with feed-forward weights (Geva et al., 2023; Meng et al., 2022).

4.3 Can we exploit this structure to control tool selection?

The PCA spectrum and the causal circuit together suggest a simple intervention: if the choice of tool is carried by a single direction in activation space, adding that direction should switch which tool gets picked. We start with the simplest case: 5 tools, all

	Peak	Depth	Importance IT	Importance Base	Importance IT/Base
✦ Gemma 3					
270M	L17	94%	1.6	0.4	4.0×
1B	L25	96%	7.4	0.6	12.3×
4B	L33	97%	10.7	3.4	3.2×
12B	L44	92%	8.5	1.6	5.3×
27B	L61	98%	14.2	1.8	7.9×
Qwen 3 / 2.5					
0.6B	L25	89%	5.9	—	
1.7B†	L20	71%	10.1	—	
4B	L35	97%	11.8	—	
8B	L35	97%	10.3	—	
14B	L39	98%	11.9	—	
2.5-7B	L27	96%	10.0	—	
∞ Llama 3.1					
8B	L31	97%	7.4	1.6	4.6×

Table 2: Attribution patching. Peak at 89–98% depth (†Qwen 1.7B outlier at 71%). Base models share the peak layer with 3–12× lower importance, except Gemma 3 1B base which peaks at L15 rather than the IT L25.

20 ordered source→target pairs, 3 held-out queries each, for 60 trials total. Steering works perfectly: **60/60 (100%; 95% CI [94–100%])** (Figure 4). To make sure this is not just any perturbation doing something weird, we try random Gaussian vectors at the same norm: 0% switch rate across 250+ trials on both Gemma 3 4B and Qwen 3 4B (Table 18). We also check that we are not overfitting to our example queries: a held-out evaluation (vectors computed from queries 1–2, tested on unseen query 3) gets the same accuracy, and resampling across 5 random splits gives $97.0\% \pm 2.4\%$.

Things get more interesting with 15 tools spanning 8 domains. Here we measure by exact match: letting the model generate five tokens after the steering intervention and checking the full tool name. Qwen 3 4B steers at **28/30 (93%)** and Gemma 3 4B at **24/30 (80%)**. Most failures involve tools that share a tokenizer prefix: four of our tools begin with `get_` (`get_weather`, `get_stock_price`, `get_news`, `get_directions`), which the tokenizer maps to the same first token. Steering correctly pushes the model into the `get_*` family (73–87% prefix-family match), but the follow-up tokens do not always resolve to the right member. When we control for this by renaming tools to have unique first tokens, accuracy recovers to the same 80–93% range with zero ambiguous cases. This suggests the gap is a tokenizer granularity issue, not a failure of the linear steering (Appendix A.1). Scaling up further, steering holds at ≥90% accuracy up to 150 tools (Table 6), with larger models handling the increase more gracefully.

The natural question is whether this works beyond synthetic tool definitions. We test on real-world APIs: τ-bench (Yao et al., 2024) airline (14 tools) and retail (16 tools), and 6 ToolBench (Qin et al., 2024) domains (10 tools each). Cross-domain ToolBench steering hits 100% on all 4B+ models except Llama 3.1 8B (90%). Base models do significantly worse (Gemma 3 4B-pt: 30%, Llama 3.1 8B-base: 50%), which suggests instruction tuning is needed for cross-domain generalization. Even on 8 deliberately ambiguous queries (like "Help me plan my trip to Paris"), steering redirects to every plausible tool at 100% (Gemma) and 96% (Qwen). Table 3 summarizes the full benchmark results; Table 15 breaks down IT vs. base.

	Steering (%) Sw-15	τ-air	τ-ret	TB-XD	Ambig.	PCA_{10}
✦ Gemma 3						
1B	43	77	53	87	47	93.6
4B	**96**	**94**	76	**100**	**100**	92.5
12B	**97**	90	80	**100**	**94**	91.2
27B	**100**	77	80	**100**	**100**	88.3
Qwen 3 / 2.5						
0.6B	50	77	47	77	69	92.5
1.7B	80	87	60	**100**	**100**	92.1
4B	**93**	80	70	**100**	**96**	90.2
8B	**100**	90	**100**	**100**	80	89.8
14B	**97**	87	63	**100**	**94**	89.7
2.5-7B	**93**	90	73	**100**	**94**	88.8
∞ Llama 3.1						
8B	83	80	**100**	90	82	89.8

Table 3: Steering accuracy (%) across benchmarks. **Bold** ≥93%. N=30 per cell. IT/base comparison in Table 15.

4.4 The model also rewrites its arguments

So steering reliably switches the tool name. But what happens to the rest of the output? Something we did not expect is that the model also *rewrites its entire argument payload* to match the new tool's schema (Table 4):

The simplest explanation is autoregression: once the target tool name is committed, the training distribution of that tool's arguments takes care of the schema. We test this with a prefill control that appends the target tool name to the prompt and lets the model autoregress the rest (Table 10, Appendix A.13). Prefill matches steering exactly on

MODEL	DIRECTION	ORIGINAL ARGS	STEERED ARGS
✦ Gemma	weather→search	"location": "Tokyo"	"query": "weather in Tokyo"
✦ Gemma	weather→translate	"location": "Tokyo"	"text": ". . . ", "lang": "en"
Qwen	translate→schedule	"text": "hello"	"date": "tomorrow"
Qwen	translate→weather	"text": "hello"	"location": "Paris"
∞ Llama	calculate→flight	"expr": "15*37"	"origin": "NYC", "dest": "LA"

Table 4: Argument schema adaptation. Steering changes the tool name *and* restructures the argument keys/values to match the target schema, without schema information in the steering vector. Full JSON outputs in Appendix A.14.

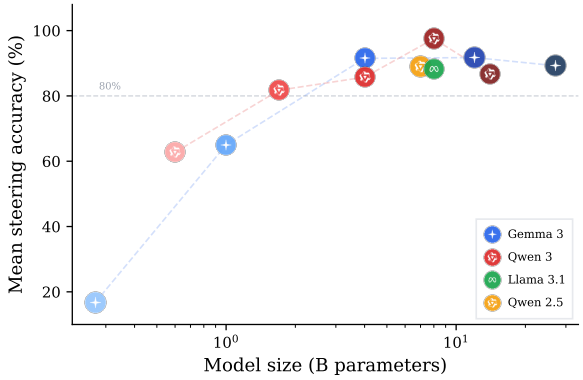

Figure 3: Mean steering accuracy vs model scale across 3 families. Accuracy rises from 17% (270M) to 89% (27B).

Gemma 3 4B and Qwen 3 4B (both 73/73 and 27/27), and on Llama 3.1 8B a 14-point gap shrinks to 4 once we restrict to pairs where steering actually switched the name. So the schema adaptation is carried by autoregressive generation, not by the steering vector itself: the vector encodes tool identity and the interface follows for free once the name is in place. The same angle explains why logit bias underperforms (it wins on only ∼20% of pairs, so most of its low schema rate comes from the name never switching).

We also probed what steering can and cannot do. *Argument steering* fails: redirecting "Tokyo" to "Paris" while keeping the same tool gives 0/30 (the tool is preserved, but the argument never changes). This makes sense. Picking a tool is choosing from a short, fixed list, so a few directions suffice. Argument values are pulled from the query on the fly, not from a fixed menu. *Format steering* (JSON vs. plain text) also fails (0/10), and *cross-family transfer* gives 0% (Gemma vectors on Qwen), suggesting each model learns its own subspace.

Multi-turn steering is the noisier of the two settings: a fixed-menu two-turn setup hits 100% at 4B+, while on τ-bench airline the matched-vs-baseline delta ranges from -30 to $+10$ pp across five 4B–14B models with no consistent direction (see Limitations).

4.5 When does this structure appear?

If the choice of tool is carried by a single direction in activation space, a natural question is whether this is always the case or whether it develops at some point. Figure 3 and Table 3 trace this with scale. At 270M the model cannot tell tools apart on the 15-tool task (27%, roughly random), and by 1B the 5-tool task is solved (100%) while 15-tool is still only partial (43%). From 4B upward all three families reach ≥83% on the 15-tool task, with Gemma 3 12B at 97% and Gemma 3 27B and Qwen 3 8B at 100%. The IT-vs-base gap is large at every scale (Cohen's $h > 0.8$; Table 15, Appendices A.6 and A.17). One anomaly is Gemma 3 27B scoring lower than 4B on the τ-bench airline task (77% vs. 94%); the CIs overlap ([59,88] vs. [84,98] at N=30), so the difference may reflect the longer context at 27B diluting the steering vector's relative strength rather than a real scaling reversal.

What changes at the feature level to explain this emergence? Sparse autoencoders (SAEs) (Lieberum et al., 2024; Rajamanoharan et al., 2024) let us count tool-specific vs. shared features at each layer. The specific-to-shared ratio sharpens with scale (≤0.3 at 270M-IT → 10.4 at 1B-IT L17 → spans 0.1–4.1 across L12–L41 at 12B-IT; Appendix A.16); base models share the geometric spectrum (89–93% top-10 variance) but at 1B L13 have 3× fewer tool-specific features and 1.5× more shared ones than their IT counterpart (Table 14, Appendix). The IT-vs-base gap in steering (Cohen's $h > 0.8$ at every scale; e.g. 65% vs. 100% at 27B) is best explained by output-layer alignment: IT directions line up 2× better with the target tool's first-token unembedding row than base directions do (§5). The 270M *base* is the surprise: it has more tool-specific features than 270M-IT (ratio 2.4 vs. 0.3) yet steering still fails (20%), so SAE counts

can decouple from causal effect, and the "sharpening" reading at 1B+ relies on activation patching, not SAE counts alone.

Two further robustness tests support this: steering holds at $\geq$90% under distractor tools up to K=150 (Table 6, Appendix A.15), and a strength sweep $\alpha \in [0.1, 3.0]$ across 15 models shows a sharp phase transition at $\alpha \approx 0.6$–0.7 (Table 16, Appendix A.18). Attribution patching closes the loop on causality: peak importance lands at 89–98% depth on all 4B+ models, and base counterparts share the peak layer but with 3–12$\times$ lower importance (Table 2).

5 Discussion

5.1 Why should we expect linearity?

The linearity conclusion rests on more than the fact that steering works. Activation patching identifies the causal components without any steering vectors, random vectors at matched norm give a 0% switch rate (so direction matters and not magnitude), and the within-topic τ-bench airline probe reads tool identity off the residual at top-1 61–89% even when all 14 tools share the airline domain, which is hard to explain as topic injection. We note that PCA dimensionality at K=15 and K=200 does not survive matched-prompt controls (Appendix A.27) and does not count as independent evidence.

Are we just injecting topics, not switching tools? The within-topic probe above is the strongest counter, and shuffled-label and Gaussian-noise sanity checks land 29–57 points below the real probe (Appendix A.30). Three further observations point the same way. The argument schemas adapt to the target tool's interface, not to the source query's topic (Table 4). The hardest failures are tools that share a tokenizer prefix despite covering completely different topics (§4.3). And the causal effect localises to attention heads rather than the MLP layers that prior work links to memorised topic associations (Geva et al., 2023; Meng et al., 2022).

The linear representation hypothesis (Park et al., 2024) predicts that discrete categories end up as directions in activation space, and this has held for binary features like sentiment and truthfulness (Tigges et al., 2023; Marks and Tegmark, 2024), though not everything is linear (Engels et al., 2024). Our contribution is that a multi-way decision (15+ tools) follows the same pattern. The unembedding matrix is itself a linear projection, so the model has a natural incentive to keep tool names linearly separated.

We verify the linearity directly in two ways. Interpolating along the steering vector, cosine similarity changes linearly and the prediction flips sharply at α≈0.6, looking like a clean linear decision boundary (Figure 16, Appendix A.19). The steering direction also aligns with the model's own first-token unembedding direction: IT models at 1B+ show cosine +0.26 to +0.29 (z=7–11 vs. random), base models roughly half (Figure 17, Appendix A.19). A causal split confirms this is the part doing the work: at matched magnitude the parallel piece alone reaches 93–100% across five models we test, while the orthogonal piece stays at 0–17%. The IT-vs-base gap in full-vector steering (97% vs. 50% on Gemma 3 4B) tracks the natural-magnitude alignment (cosine 0.26 vs. 0.13).

5.2 Catching mistakes before they happen

The same per-tool means double as a before-execution monitor. For a new query we compute cosine to each tool mean and rank the candidates, and a small top-1/top-2 gap flags that the model is unsure (Appendix A.26). On BFCL v3 (Yan et al., 2024) the base-vs-IT split is striking (Table 23, Appendix A.26). IT generation already hits 90–95%, so readout adds nothing on top. On base models the relationship reverses: greedy generation lands at 2–10% while readout recovers 61–82%, a gain of +56 to +72 points. One reading is that pretraining encodes tool identity before instruction tuning wires it to output. The same gap also flags errors: on Gemma 3 12B the smallest-gap quartile has a 14% error rate while the largest gap quartile has 0%, and 27B shows a 21$\times$ concentration ratio. Matched-prompt cosines complicate this: IT tool/topic pairs land within $\Delta \leq 0.036$ on Gemma (Qwen and Llama IT spread the other way, base is much tighter at cos $\sim$0.999), so the readout works through argmax over small differences and we treat the IT-vs-base reading as suggestive (Table 24, Appendix A.26).

5.3 Conclusion

In single-turn, fixed-menu settings, the choice of tool inside an LLM is carried by a single direction in activation space, with one direction per pair of tools. Across three families and 270M–27B, adding that direction switches the model's tool choice at 83–100% on 4B+ instruction-tuned models (93–100% on Gemma 3 and Qwen 3).

Limitations

There are clear limits to what we have shown, and being upfront about them matters for anyone who might want to build on this work. The linear structure only appears above 270M parameters; sub-1B models do not have exploitable structure (Section 4.5). Steering breaks down when tools are near-synonymous (0% on 10 similar Sports APIs; Appendix A.4), which is really a limit of the model itself: if the model cannot distinguish two tools, neither can we. Tool identity is linear but argument content is not: steering can change *which* tool gets called but not *what values* get passed to it (Section 4.3). Steered outputs match the target schema (Table 4) but sometimes fabricate arguments (like "Pizza Palace" for a restaurant name). A real API server would likely reject such inputs. Our evaluation measures whether the model picks the right tool and produces the right JSON structure, not whether the arguments would actually work in production. Steering should be viewed as an analytical and routing tool (as in cosine select), not as a standalone method for end-to-end API execution; measuring actual task completion is the natural next step.

Multi-turn behaviour is mixed. On a synthetic two-turn setup with a fixed menu, applying single-turn vectors at the second turn gives 100% at 4B+. On a realistic agent benchmark (τ-bench airline), comparing against fresh, non-crashing baselines gives a much smaller and model-dependent effect: the matched-vs-baseline delta ranges from -30 to $+10$ percentage points across five 4B–14B models (Gemma 3 4B / 12B, Qwen 3 4B / 8B / 14B). Gemma 12B L47 and Qwen 8B show the largest drops. We read this as the linear steering signal being reliable in single-turn, fixed-menu settings but competing for residual-stream capacity, or interacting badly with accumulated tool outputs, once the conversation carries substantial history. Longer multi-turn chains beyond this remain untested, and closing this gap is the main practical follow-up.

There is also a practical limit on prompt length. When tool definitions include descriptions, the prompt grows to 16K tokens at K=750, and at that point Gemma 3 4B's per-tool representations start to blur together (k_{90} drops from 57 to 27). Without descriptions, the same model handles K=1500 tools smoothly (k_{90} grows to 116 with no collapse), because the prompt stays shorter (8K tokens). The bottleneck is not the number of tools but the total prompt length. We see the same pattern with tool descriptions in steering (Appendix A.24): instruction-tuned models at 4B+ already reach 93–100% without descriptions and gain at most a few points from them, while smaller models (Qwen 0.6B) and base models get *worse*, because the longer prompt overwhelms their ability to focus on the right parts. This limit also depends on architecture: at K=500 with descriptions, 8-head models (Gemma 3 4B) need 57 dimensions, 16-head models (Gemma 3 12B) need 129, and 32-head models (Qwen) need 147–151 (Appendix 32). More heads and more parameters both buy stability: at K=750 ($\sim$16K tokens), Gemma 3 4B's k_{90} drops to 27 and does not recover at K=1000/1500, but Gemma 3 12B and 27B dip only briefly (131$\rightarrow$130 at K=500$\rightarrow$1000 for 27B) and fully recover their compression (Figure 7). Scale alone extends the effective context: 32-head models show no collapse even at K=750, and larger Gemma models (12B/27B) tolerate at least K=1000 before prompt length outgrows the context budget. All three families we test (Gemma 3, Qwen 3 / 2.5, Llama 3.1) are dense transformers; we do not know if this extends to mixture-of-experts or state-space models.

The unembedding decomposition (Section 5, Appendix A.19) anchors on the target tool's *first*-token unembedding row, so for tools that share a tokenizer prefix (`get_weather`, `get_stock_price`, `get_news`, `get_directions`) the parallel direction is partly the shared `get_` direction rather than the tool-specific suffix tokens (`_weather` vs. `_stock_price`). A sum or mean over the full tool-name tokens is the natural extension, which we leave for follow-up.

Ethics Statement

Reproducibility. All models are publicly available on HuggingFace (Table 19). SAEs are from Gemma Scope 2 (Lieberum et al., 2024) and NeuronPedia (Lin, 2024). We use TransformerLens (Nanda and Bloom, 2022) via SAELens (Bloom et al., 2024) for activation caching and steering. Steering vectors are computed from 2–3 queries per tool with no training. For the 27B model, we verified results using HuggingFace `transformers` with `device_map=auto`. Code and data are attached to this submission as supplementary archives (Software and Data).

Dual-use and defense. Our steering method redirects a model's tool selection at inference time, which is dual-use: an operator with white-box access to an open-weight model could in principle silently reroute consequential tool calls without leaving an output-level trace. We see the net effect of this work as defender-side. The same per-tool mean activations that enable steering also flag likely-wrong calls (Section 5), and the mechanistic visibility lets evaluators audit which components carry tool selection before trusting an agent with consequential actions. All models we study are publicly available on HuggingFace, so the work does not widen access to any system; output-level schema validation and allowlisting provide an independent defense against redirects, whether malicious or accidental.

References

Emmanuel Ameisen and 1 others. 2025. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*.

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. 2024. Refusal in language models is mediated by a single direction. *arXiv preprint arXiv:2406.11717*.

Joseph Bloom and 1 others. 2024. SAELens: A library for sparse autoencoder training, analysis, and interpretability.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, and 1 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. In *NeurIPS*.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2023. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, and 1 others. 2022. Toy models of superposition. *Transformer Circuits Thread*.

Joshua Engels, Eric J Michaud, Isaac Liao, Wes Gurnee, and Max Tegmark. 2024. Not all language model features are one-dimensionally linear. *arXiv preprint arXiv:2405.14860*.

Gemma Team. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Mor Geva, Jaap Bastings, Katja Filippova, and Amir Globerson. 2023. Dissecting recall of factual associations in auto-regressive language models. *arXiv preprint arXiv:2304.14767*.

Aaron Grattafiori and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. *Advances in Neural Information Processing Systems*.

Kait Healy, Bharathi Srinivasan, Visakh Madathil, and Jing Wu. 2026. Internal representations as indicators of hallucinations in agent tool selection. *arXiv preprint arXiv:2601.05214*.

Dan Hendrycks, Mantas Mazeika, and Thomas Woodside. 2023. An overview of catastrophic AI risks. *arXiv preprint arXiv:2306.12001*.

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

Minghao Li and 1 others. 2023b. API-Bank: A comprehensive benchmark for tool-augmented LLMs. *arXiv preprint arXiv:2304.08244*.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. 2024. Gemma scope: Open sparse autoencoders everywhere all at once on Gemma 2. *arXiv preprint arXiv:2408.05147*.

Johnny Lin. 2024. NeuronPedia: A platform for mechanistic interpretability.

Samuel Marks and Max Tegmark. 2024. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. *arXiv preprint arXiv:2310.06824*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *NeurIPS*.

Neel Nanda and Joseph Bloom. 2022. TransformerLens: A library for mechanistic interpretability of GPT-style language models.

Neel Nanda, Andrew Lee, and Martin Wattenberg. 2023. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*.

Kiho Park, Yo Joong Choe, and Victor Veitch. 2024. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. 2024. Jumping ahead: Improving reconstruction fidelity with JumpReLU sparse autoencoders. *arXiv preprint arXiv:2407.14435*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, and 1 others. 2024. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. *Transformer Circuits Thread*.

Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. 2023. Linear representations of sentiment in large language models. *arXiv preprint arXiv:2310.15154*.

Eric Todd, Millicent L Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. 2024. Function vectors in large language models. *arXiv preprint arXiv:2310.15213*.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. 2024. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Simas Sakenis, Jason Huang, Yaron Singer, and Stuart Shieber. 2020. Causal mediation analysis for interpreting neural NLP: The case of gender bias. *arXiv preprint arXiv:2004.12265*.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. *arXiv preprint arXiv:2211.00593*.

Youjin Wang, Run Zhou, Rong Fu, Shuaishuai Cao, Hongwei Zeng, Jiaxuan Lu, Sicheng Fan, Jiaqiao Zhao, and Liangming Pan. 2026. ASA: Training-free representation engineering for tool-calling agents. *arXiv preprint arXiv:2602.04935*.

Fanjia Yan, Huanzhi Mao, Charlie Ji, Shishir Patil, Ion Stoica, Joseph E Gonzalez, and Hao Zhang. 2024. Berkeley function calling leaderboard. In *NeurIPS*.

Shunyu Yao and 1 others. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, and 1 others. 2023. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*.

A.1 Failure mode analysis

Steering failures fall into three categories.

Prefix collisions. Four tools (`get_weather`, `get_stock_price`, `get_news`, `get_directions`) share the first token `get`. Steering reliably pushes the model to the correct prefix family, but the disambiguation tokens (`_weather` vs. `_stock_price`) do not always follow: exact-match accuracy for within-`get_*` pairs is lower than for unique-prefix tools. This is a tokenization granularity issue, not a failure of the linear steering signal.

Semantic resistance. The model reverts to the original tool when the query is highly specific. "Calculate 15 * 37" resists steering away from `calculate`, for example.

Near-synonyms. Semantically overlapping tools like the 10 similar Sports APIs give 0% steering. No failures involve hallucinated tool names or malformed JSON.

A.2 Tool definitions

The 15 tools used in our experiments span diverse domains: information retrieval (get_weather, search, get_stock_price, get_news), computation (calculate, convert_currency), communication (send_email, create_note), scheduling (schedule, set_alarm, book_flight), navigation (get_directions), entertainment (play_music), food ordering (order_food), and language (translate).

A.3 Full pairwise switching matrix (5 tools)

All 20 source→target directions hit 100% on held-out queries across the three families, with no failures at this scale.

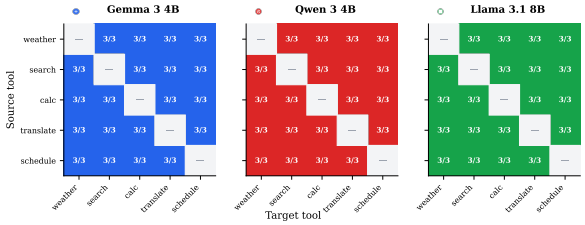

Figure 4: 5-tool pairwise switching matrix. All 20 pairs hit 100% on held-out queries for every model and family.

A.4 Cross-benchmark and ToolBench analysis

Figure 5 shows steering accuracy across 12 models and 4 benchmarks; Figure 6 shows ToolBench per-domain breakdown on Gemma 3 4B.

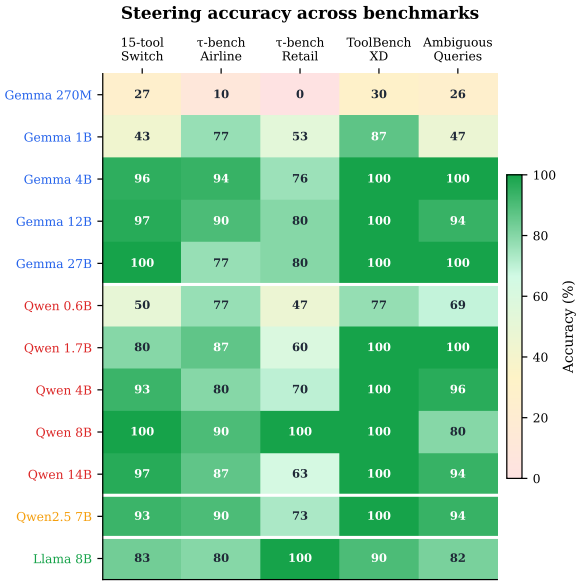

Figure 5: Steering accuracy across 12 models × 4 benchmarks.

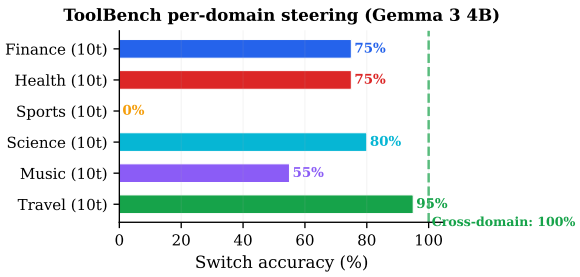

Figure 6: ToolBench per-domain steering (Gemma 3 4B). Sports fails due to near-synonymous tools.

A.5 NeuronPedia cross-reference

We cross-referenced our findings with independently labeled features on NeuronPedia[1] for Gemma 3 4B-IT. The following pre-existing feature labels align with our discovered circuit stages (Table 5):

LAYER	SAE	FEATURE	NEURONPEDIA LABEL
2	tc-262k	159677	"tool description"
9	tc-262k	73435	"tool selection"
9	res-16k	5625	"function calls"
16	tc-262k	177725	"tool calls"
17	res-16k	1645	"function calls"
17	res-16k	15177	"tools and their functions"
22	tc-262k	19924	"tool use invocation"
22	res-16k	3122	"function calls"
25	tc-262k	540	"JSON opening brace"
29	res-262k	95374	"tool selection"
29	res-16k	9781	"JSON structured output"

Table 5: NeuronPedia feature labels for Gemma 3 4B-IT. Labels were assigned independently by automated methods. The layer progression matches our three-stage circuit.

The top activating examples for these features are consistently about code generation, API calls, and structured outputs, which fits with the idea that tool-calling piggybacks on general programming circuitry. The three features we identified via SAE differential analysis (Section 4.2), Layer 17 F#201 (tool intent), F#489 (tool context), and F#359 (direct answer), do not yet have auto-generated NeuronPedia descriptions, but their top activating examples are also coding tasks.

A.6 Scaling results

Tool switching accuracy as a function of the number of tools:

K	G-4B	G-12B	G-27B	Q-4B	Q-8B	Q-14B	Q2.5-7B	L-8B
5	100	100	100	100	75	100	100	100
15	100	100	100	100	100	100	95	95
30	100	100	100	100	95	100	90	100
50	95	100	95	100	100	100	100	100
75	95	100	95	100	100	100	100	100
100	100	100	100	100	100	95	95	100
150	90	100	100	100	100	95	100	100

Table 6: K-tool scaling: switch accuracy (%) among randomly sampled tools. G=Gemma, Q=Qwen, Q2.5=Qwen 2.5, L=Llama.

A.7 PCA method details

For each of K tools, we collect 2–3 example queries and cache the residual stream activation

[1] https://www.neuronpedia.org/gemma-3-4b-it

at the last token position in the penultimate layer. We average these to get one mean vector per tool: $\bar{\mathbf{h}}_i \in \mathbb{R}^d$ for $i = 1, \ldots, K$.

Step 1: Center. Subtract the grand mean so the data is zero-centered: $\mathbf{C}_i = \bar{\mathbf{h}}_i - \frac{1}{K}\sum_j \bar{\mathbf{h}}_j$

Step 2: SVD. Stack the centered vectors into a $K \times d$ matrix and compute the singular value decomposition $\mathbf{C} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$, where the diagonal entries $s_1 \geq s_2 \geq \ldots$ are the singular values.

Step 3: Variance explained. The fraction of total variance captured by the first k components is: $\mathrm{Var}(k) = \frac{\sum_{i=1}^{k} s_i^2}{\sum_{i=1}^{K-1} s_i^2}$

We define k_{90} as the smallest k such that $\mathrm{Var}(k) \geq 0.90$.

Interpretation. If k_{90} is much smaller than $K-1$ (the theoretical maximum number of independent directions for K points), then the tools are not spread uniformly across the space but cluster into a lower-dimensional subspace. In our experiments, $k_{90} = 10$ for 15 controlled tools (max 14), and $k_{90} = 57$ for 500 ToolBench APIs (max 499). A Monte Carlo baseline (random points in $\mathbb{R}^{2560}$) gives $k_{90} = 392$ for 500 points, which supports reading the compression as a real effect rather than a geometric inevitability.

A.8 PCA dimensionality vs. tool count

See Table 31 in Appendix A.31 for the full scaling table and additional analysis (cross-model comparison, sampling sensitivity).

A.9 Error correction

At 100 tools, the model makes 15 baseline errors (85% accuracy). We can correct 9 of those 15 (60%) by steering from the wrong tool toward the right one. More tellingly, the model's internal cosine similarity already ranks the correct tool at position 1 in 80% of error cases. This suggests that most selection errors happen at the decoding level, not the representation level: the model "knows" the right tool internally but picks the wrong token.

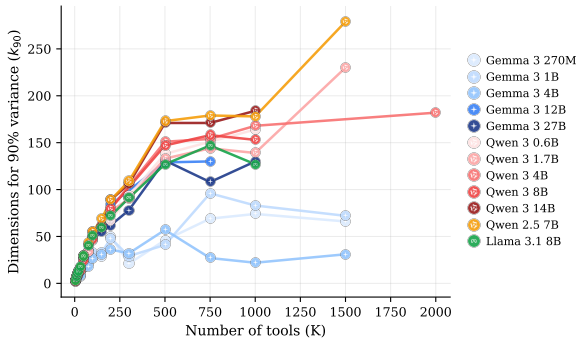

Figure 7: k_{90} as the number of real ToolBench APIs grows from 5 to 2,000, across 12 instruction-tuned models from three families (Gemma 3, Qwen 3 / Qwen 2.5, Llama 3.1; stratified sampling across 49 domains, with descriptions). All models compress tool activations into k_{90} that grows far slower than K, but the absolute level depends more on architecture (attention heads) than parameter count: the Gemma 3 family (blue shades, 8 heads) stays below $k_{90}\approx100$ across all sizes, while Qwen and Llama (16–32 heads) use 2–3$\times$ more directions. Gemma 3 4B's k_{90} also collapses past K=500 as the prompt outgrows its effective context. Qwen 2.5 7B scales furthest (k_{90}=279 at K=1500).

A.10 Cross-model circuit comparison

PROPERTY	GEMMA 3 4B	QWEN 3 4B	LLAMA 3.1 8B
Layers / d_{model}	34 / 2560	36 / 2560	32 / 4096
Steering effective	L17–L33	L23–L35	L31
PCA top-10	92.5%	90.2%	89.8%
ToolBench XD	100%	100%	90%
SAE source	Scope 2 (4L)	mwhanna tc (36L)	Goodfire L19
Larger models (ToolBench cross-domain / PCA):			
Gemma 3 12B (48L)		100% / 91.1%	
Gemma 3 27B (62L)		100% / 88.3%	
Qwen 3 14B (40L)		100% / 89.7%	

Table 7: Cross-architecture comparison. All three families show 88–93% PCA variance in 10 dimensions.

A.11 Cross-model PCA comparison

Table 8 reports cumulative variance and k_{90} across Gemma and Qwen at K=15.

TOP-k PCS	GEMMA 3 4B	QWEN 3 4B
1	27.5%	21.0%
3	51.3%	48.6%
5	67.5%	64.5%
10	92.5%	90.2%
14	100%	100%

Table 8: Cumulative variance explained by top-k principal components of 15-tool mean activations. Both models show remarkably similar subspace dimensionality despite independent training.

A.12 Cross-layer steering (Qwen 3)

Steering effectiveness by layer for Qwen 3 4B (get_weather → search, 5-pair average; Table 9):

Layer	Separation	Switch Success	Multi-pair Acc.
L0	0.3	×	0/5
L9	3.9	×	0/5
L18	12.1	×	0/5
L23	38.3	✓	–
L27	74.8	✓	5/5
L35	252.8	✓	5/5

Table 9: Qwen 3 4B cross-layer steering. Fails at L0–L18; works at L23+ once representations diverge.

A.13 Schema-correctness controls

	A: base src schema	B: prefill tgt schema	C: steer tgt schema
Gemma 3 4B IT	87%	73%	73%
Qwen 3 4B	60%	27%	27%
Llama 3.1 8B IT	90%	57%	43%

Table 10: Schema-correct rate across three conditions on the same 30 source→target pairs. A: no intervention (expect source schema). B: prompt is prefilled with the target tool name and the model autoregresses the arguments. C: activation steering replaces B's prefill. B and C land at the same rate on Gemma 3 4B and Qwen 3 4B; on Llama 3.1 8B the 14-point gap shrinks to 4 points once we restrict to pairs where steering actually switched the tool name, so the remaining difference reflects commit strength, not schema information in the steering vector.

A.14 Full generation examples

All outputs are generated autoregressively from the prefix `{"tool": "` with no post-processing. The steering vector is applied only at the first forward pass (Table 11; ambiguous-query variant in Table 12).

Model	Direction	Original Args	Steered Args
+	weather→search	`"location": "Tokyo"`	`"query": "best time to visit Paris"`
+	weather→translate	`"location": "Tokyo"`	`"text": ". . . ", "lang": "ja"`
+	weather→flight	`"location": "Tokyo"`	`"origin": "NYC", "dest": "Paris"`
+	weather→email	`"location": "Tokyo"`	`"to": "colleague@. . ."`
+	weather→note	`"location": "Tokyo"`	`"title": "Meeting Prep"`
+	weather→food	`"location": "Tokyo"`	`"restaurant": "Pizza Palace"`
🦙	translate→weather	`"text": "hello"`	`"location": "Paris"`
🦙	weather→schedule	`"location": "Tokyo"`	`"date": "tomorrow"`
🦙	search→currency	`"query": "AI news"`	`"amount": "1000", "from": "USD"`
∞	calculate→flight	`"expr": "15*37"`	`"origin": "NYC", "dest": "LA"`
∞	weather→search	`"location": "Tokyo"`	`"query": "weather in Tokyo"`

Table 11: Steered generation examples across three families. Every output restructures its arguments.

Query	Steered To	Steered Args
"Plan my trip to Paris"	book_flight	`"origin": "NYC", "dest": "Paris"`
	search	`"query": "best time to visit Paris"`
	get_directions	`"location": "Paris"`
"Prepare for meeting"	schedule	`"date": "tomorrow"`
	create_note	`"title": "Meeting Prep"`
	send_email	`"to": "colleague@. . ."`
"What about Tokyo?"	search	`"query": "Tokyo travel guide"`
	translate	`"text": ". . . ", "lang": "ja"`
	get_news	`"query": "Tokyo"`

Table 12: Ambiguous query steering (Gemma 3 4B). All default to `get_weather`; steering redirects to every plausible target with appropriate args. 100% (Gemma), 96% (Qwen).

A.15 Steering strength and tool-count scaling

Figure 8 shows the α phase transition; Figure 9 shows K-tool scaling across model sizes.

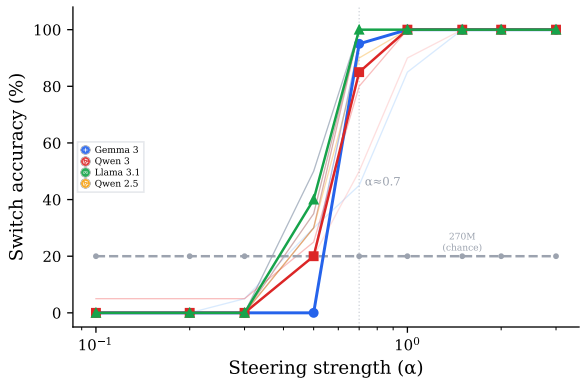

Figure 8: Phase transition at $\alpha \approx 0.7$. 4B+ models jump to 95–100% with no collapse.

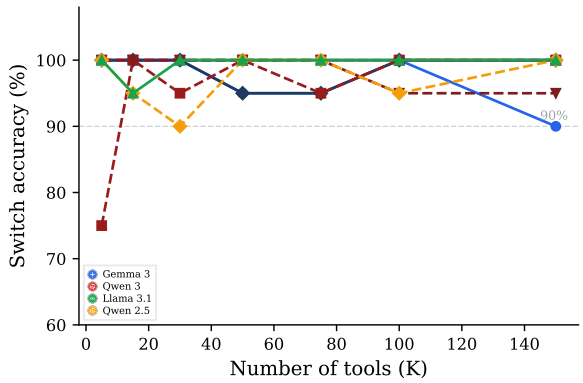

Figure 9: K-tool scaling. Larger models sustain accuracy better.

A.16 Full SAE feature analysis

Table 13 reports per-layer SAE feature counts across all model scales and families, and Figure 10 visualizes the same data.

Model	Layer	Depth	Feature counts Active	Shared	Specific	Sp/Sh Ratio
Gemma 3 270M-IT	L5	28%	34	30	0	0.0
Gemma 3 270M-IT	L9	50%	51	40	9	0.2
Gemma 3 270M-IT	L15	83%	67	51	16	0.3
Gemma 3 270M-base	L5	28%	—	41	1	0.0
Gemma 3 270M-base	L15	83%	—	32	77	2.4
Gemma 3 1B-IT	L7	27%	48	41	3	0.1
Gemma 3 1B-IT	L13	50%	56	27	36	1.3
Gemma 3 1B-IT	L17	65%	67	12	125	10.4
Gemma 3 1B-IT	L22	85%	51	13	81	6.2
Gemma 3 1B-base	L13	50%	55	40	11	0.3
Gemma 3 1B-base	L22	85%	69	33	43	1.3
Gemma 3 4B-IT	L0–3	9%	*N/A (transcoders)*	38		—
Gemma 3 4B-IT	L17	50%	66	31	50	1.6
Gemma 3 4B-base	L17	50%	68	31	49	1.6
Gemma 3 4B-base	L22	65%	80	35	73	2.1
Gemma 3 12B-IT	L12	25%	44	37	3	0.1
Gemma 3 12B-IT	L24	50%	48	20	31	1.6
Gemma 3 12B-IT	L31	65%	52	24	56	2.3
Gemma 3 12B-IT	L41	85%	49	19	78	4.1
Gemma 3 12B-base	L12	25%	60	38	10	0.3
Gemma 3 12B-base	L31	65%	77	30	62	2.1
Gemma 3 12B-base	L41	85%	78	33	78	2.4
Gemma 3 27B-IT	L16	26%	47	43	4	0.1
Llama 3.1 8B-IT*	L19	59%	85	14	95	6.8

Table 13: Full SAE feature analysis across all scales and layers. *Goodfire (`goodfire.ai`; 65K vs 16K Scope).

SAE feature sharpening

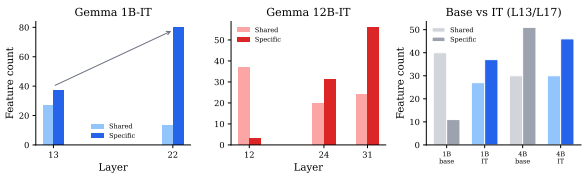

Figure 10: SAE feature sharpening across layers. Specific features increase and shared features decrease with depth.

A.17 Base vs instruction-tuned comparison

Figure 11 compares base vs instruction-tuned steering with Cohen's h effect sizes; Figure 12 shows the attribution-patching peak layer across families.

Scale	Switch-5 IT	Switch-5 Base	ToolBench XD IT	ToolBench XD Base	Cohen's h
Gemma 3 1B	100	20	87	27	2.21
Gemma 3 4B	100	55	100	30	1.47
Gemma 3 12B	95	70	100	33	0.71
Gemma 3 27B	100	65	100	90	1.27
Llama 3.1 8B	100	50	90	50	1.57

Table 14: Base vs instruction-tuned models (%). Effect sizes are large ($h \geq 0.71$; all exceeding Cohen's "medium" threshold of 0.5). IT consistently outperforms base; the gap narrows with scale but persists even at 27B.

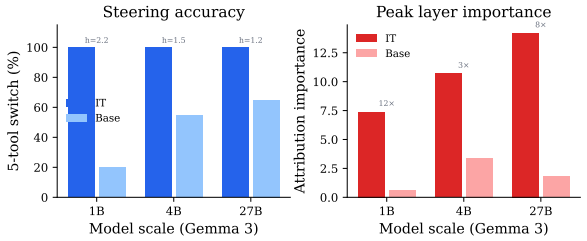

Figure 11: Base vs IT. Left: 5-tool accuracy (base improves with scale but never matches IT). Right: attribution importance (IT is 3–12× higher).

	Sw-15	τ-Air	τ-Ret	TB-XD
	IT / base where available			
✦ Gemma 3				
270M	27 / 27	$10^{\dagger}$ / $10^{\dagger}$	$0^{\dagger}$ / $0^{\dagger}$	30 / 30
1B	43 / 27	77 / $10^{\dagger}$	53 / $0^{\dagger}$	87 / 27
4B	**96** / 37	**94** / 60	76 / 60	**100** / 30
12B	**97** / 53	90 / 53	80 / **93**	**100** / 33
27B	**100** / 70	77 / **97**	80 / 83	**100** / 90
Qwen 3 / 2.5				
0.6B	50	77	47	77
1.7B	80	87	60	**100**
4B	**93**	80	70	**100**
8B	**100**	90	**100**	**100**
14B	**97**	87	63	**100**
2.5-7B	**93**	90	73	**100**
∞ Llama 3.1				
8B	83 / 53	80 / 80	**100** / **100**	90 / 50

Table 15: Per-cell steering accuracy (%, prefix match) across all 12 models, with IT/base split where both variants are tested. No public Qwen base variants. All $p<0.001$ except † (n.s.). **Bold** $\geq$93%. N=30 per cell. CIs in Table 18. Body-level summary in Table 3 and Section 4.5.

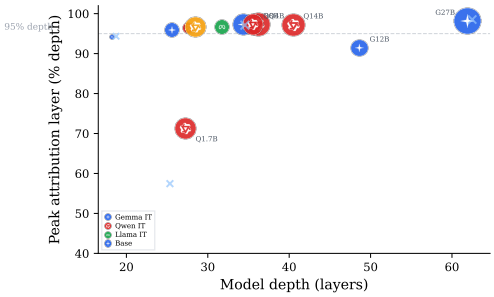

Figure 12: Attribution patching peak layer. IT peaks at 92–98% depth; base models share the location but with much lower importance.

Attribution peak depth

A.18 Steering strength calibration

MODEL	α=0.3	α=0.5	α=0.7	α=1.0	α=2.0	α=3.0	TRANS.
Gemma 3 270M	20%	20%	20%	20%	20%	20%	—
Gemma 3 1B	5%	30%	45%	85%	100%	100%	∼1.0
Gemma 3 4B	0%	0%	95%	100%	100%	100%	0.7
Gemma 3 12B	0%	20%	95%	100%	100%	100%	0.7
Gemma 3 27B	0%	50%	100%	100%	100%	100%	0.7
Qwen 3 0.6B	0%	20%	50%	90%	100%	100%	∼1.0
Qwen 3 1.7B	5%	25%	80%	100%	100%	100%	0.7
Qwen 3 4B	0%	20%	85%	100%	100%	100%	0.7
Qwen 3 8B	0%	30%	100%	100%	100%	100%	0.7
Qwen 3 14B	0%	35%	100%	100%	100%	100%	0.7
Qwen 2.5 7B	0%	35%	85%	100%	100%	100%	0.7
Llama 3.1 8B	0%	40%	100%	100%	100%	100%	0.7
Base (pre-trained) models:							
Gemma 3 270M-base	20%	20%	20%	20%	25%	35%	—
Gemma 3 1B-base	20%	20%	20%	20%	40%	45%	—
Gemma 3 4B-base	5%	30%	55%	80%	75%	65%	∼0.7

Table 16: Steering strength calibration. "Trans." = α at 50% accuracy. 4B+ IT models transition sharply at $\alpha \approx 0.7$; base models are flat or collapse at high α.

A.19 Cross-layer divergence and attention

Figure 13 shows cross-tool divergence by layer depth, with steering failing below the inflection point; Figure 14 shows attention from the output position to tool-name and entity tokens (Gemma 3 4B). Figure 15 additionally shows that synthetic, τ-bench, and ToolBench tools occupy the same low-dimensional subspace.

Cross-source tool subspace

Linearity interpolation (all pairs)

Unembedding decomposition of the steering vector

We split each steering vector $d = \bar{h}_b - \bar{h}_a$ into a piece parallel to the target tool's first-token unembedding row and an orthogonal remainder, and steer with each alone. The parallel piece is only 11–17% of the full norm, so we rescale both pieces

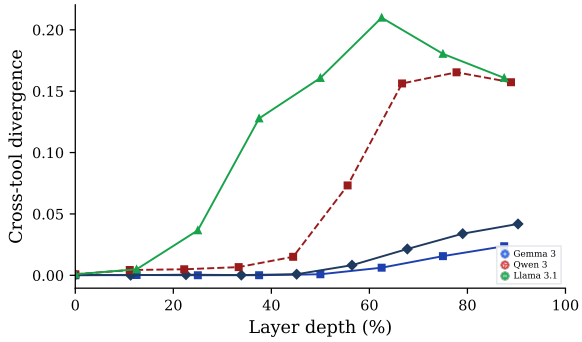

Figure 13: Cross-tool divergence by layer depth. Divergence rises sharply in the final 20–30% of layers. Steering fails below the inflection point.

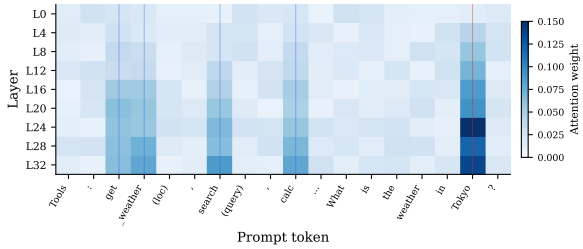

Figure 14: Attention from output position to prompt tokens (Gemma 3 4B). Tool-name and entity tokens receive increasing attention at deeper layers.

to the full vector's L2 norm ("par*" and "orth*") to separate direction from magnitude (Table 17).

MODEL	FULL	PAR	PAR*	ORTH	ORTH*	ZERO
Gemma 3 4B IT	97%	3%	**100%**	7%	7%	0%
Gemma 3 4B base	50%	3%	**100%**	7%	7%	0%
Qwen 3 4B	93%	0%	**100%**	0%	0%	0%
Llama 3.1 8B IT	77%	3%	**93%**	17%	17%	0%
Llama 3.1 8B base	77%	0%	**100%**	7%	7%	0%

Table 17: Decomposing the steering vector against the target tool's first-token unembedding direction, on 30 source→target pairs. Full: the original $\bar{h}_b - \bar{h}_a$ at α=0.7. Par and Orth: natural parallel and orthogonal components of d (each at its own norm). Par* and Orth*: the same components rescaled to match the full vector's L2 norm.

Both base and IT par* reach ≥93% at matched magnitude, so this test alone cannot pin the IT-vs-base gap on the parallel direction. The full-vector gap (Gemma 3 4B: 97% IT vs. 50% base at α=0.7) tracks the natural-magnitude alignment (IT cosine ∼0.26, base ∼0.13), consistent with alignment being the quantity that matters.

A.20 Developer tool steering

We also test on software development tools (10 tools like `create_file`, `run_tests`, `git_commit`,

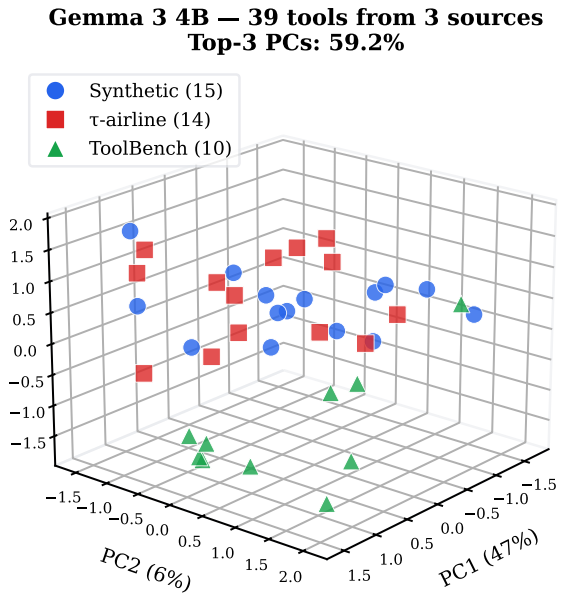

Figure 15: 3D PCA of 39 tools from 3 sources (15 synthetic, 14 τ-bench airline, 10 ToolBench) in Gemma 3 4B. All tools share the same low-dimensional subspace regardless of source, suggesting linearity extends to real-world APIs.

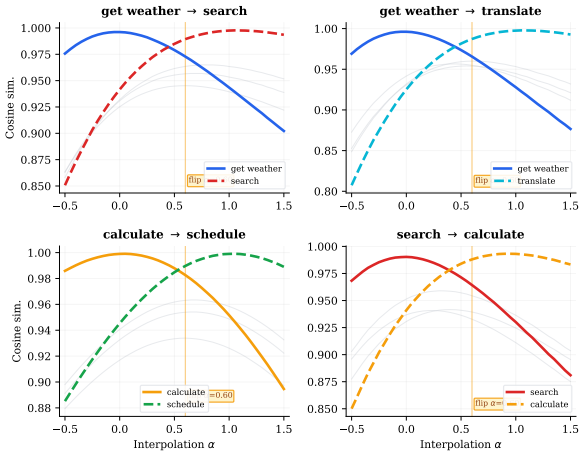

Figure 16: Linearity interpolation for all 4 tested tool pairs. In every case, cosine similarity changes smoothly and the prediction flips sharply at $\alpha \approx 0.6$–0.7, consistent with linear tool selection across diverse tool combinations.

deploy, etc.), which share much more semantic overlap than the primary 15-tool set. Gemma 3 4B gets 63% here, lower than the 73–87% on general tools, which makes sense given the overlap.

```
Original: {"tool": "create_file", "path": "src/config.yaml",
"content": . . . }
Steered:  {"tool": "run_tests", "suite": "unit", "path":
"src/"}
```

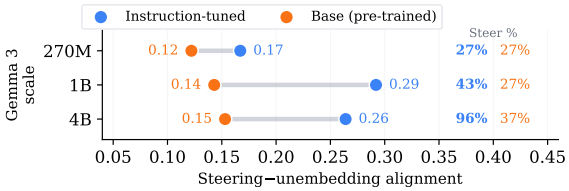

Figure 17: Steering vectors align with unembedding directions for the target tool. IT models show 2× higher alignment than base, and z-scores increase with scale.

```
Original: {"tool": "deploy", "environment": "staging",
"version": . . . }
Steered:  {"tool": "git_commit", "message": "Deploy staging
v2.1"}
```

PCA top-5 variance is 82.0% (vs. 67.5% for general tools) and top-9 hits 100%, meaning the developer tools are packed into an even smaller subspace. The takeaway: the more semantically similar the tools, the harder they are to steer apart.

A.21 Full confidence intervals

Model	Sw-15	τ-Air	τ-Ret	TB-XD
Gemma 3 270M	27 [14,44]	10 [3,26]	0 [0,11]	30 [17,48]
Gemma 3 1B	43 [27,61]	77 [59,88]	53 [36,70]	87 [70,95]
Gemma 3 4B	96 [89,98]	94 [84,98]	76 [63,86]	100 [89,100]
Gemma 3 12B	97 [83,99]	90 [74,97]	80 [63,90]	100 [89,100]
Gemma 3 27B	100 [89,100]	77 [59,88]	80 [63,90]	100 [89,100]
Qwen 3 0.6B	50 [33,67]	77 [59,88]	47 [30,64]	77 [59,88]
Qwen 3 1.7B	80 [63,90]	87 [70,95]	60 [42,75]	100 [89,100]
Qwen 3 4B	93 [79,98]	80 [63,90]	70 [52,83]	100 [89,100]
Qwen 3 8B	100 [89,100]	90 [74,97]	100 [89,100]	100 [89,100]
Qwen 3 14B	97 [83,99]	87 [70,95]	63 [46,78]	100 [89,100]
Llama 3.1 8B	83 [66,93]	80 [63,90]	100 [89,100]	90 [74,97]
Qwen 2.5 7B	93 [79,98]	90 [74,97]	73 [56,86]	100 [89,100]

Table 18: Wilson 95% CIs for all models and benchmarks. N=30 per cell. Format: accuracy [%] [lower, upper].

A.22 Models and resources

Table 19 lists all models used in this work. Table 20 lists SAE/transcoder sources and software tools.

Family	Model	Params	HuggingFace ID
Gemma 3	Gemma 3 270M IT	270M	`google/gemma-3-270m-it`
	Gemma 3 1B IT	1B	`google/gemma-3-1b-it`
	Gemma 3 4B IT	4B	`google/gemma-3-4b-it`
	Gemma 3 12B IT	12B	`google/gemma-3-12b-it`
	Gemma 3 27B IT	27B	`google/gemma-3-27b-it`
	Gemma 3 1B/4B/27B PT	—	`google/gemma-3-{1b,4b,27b}-pt`
Qwen 3	Qwen 3 0.6B	0.6B	`Qwen/Qwen3-0.6B`
	Qwen 3 1.7B	1.7B	`Qwen/Qwen3-1.7B`
	Qwen 3 4B	4B	`Qwen/Qwen3-4B`
	Qwen 3 8B	8B	`Qwen/Qwen3-8B`
	Qwen 3 14B	14B	`Qwen/Qwen3-14B`
Qwen 2.5	Qwen 2.5 7B Instruct	7B	`Qwen/Qwen2.5-7B-Instruct`
Llama 3.1	Llama 3.1 8B Instruct	8B	`meta-llama/Llama-3.1-8B-Instruct`

Table 19: All models used in this work. PT = pre-trained (base); all others are instruction-tuned.

Resource	Coverage	Source
Gemma Scope 2 SAEs	Gemma 3 (L9/17/22/29)	`gemma-scope-2-4b-it-res`
Gemma Scope 2 Transcoders	Gemma 3 (all 34 layers)	Ameisen et al. (2025)
NeuronPedia SAEs	Qwen 3 (all layers)	`neuronpedia.org`
Goodfire SAEs	Llama 3.1 8B (L19)	`goodfire.ai`
TransformerLens	Activation caching	v2.17+ (Nanda and Bloom, 2022)
SAELens	SAE loading/analysis	v6.37+ (Bloom et al., 2024)
Circuit-tracer	Attribution graphs	Ameisen et al. (2025)

Table 20: SAE/transcoder sources and software tools.

A.23 Cross-lingual tool space alignment

If the tool subspace encodes intent rather than surface language, then English and Chinese queries for the same tool should land in the same region of activation space. We test this by querying 5 tools in 8 languages (English, Chinese, Japanese, French, Spanish, German, Korean, Arabic), keeping tool definitions always in English. For each tool and language, we compute a mean activation from 1 query and evaluate on 3 held-out queries.

Steering across languages. The strongest result is causal: steering vectors computed entirely from English queries successfully switch tool selection on non-English queries. On Qwen 3 4B, English vectors achieve 100% accuracy on Chinese, Japanese, French, and Spanish queries (10 pairs each, same pairs across all languages). On Gemma 3 4B, the rate is 90–100%. This means a system could compute its steering vectors in one language and apply them to users in any language.

How much of the similarity is tool-specific? Same-tool cross-lingual cosine similarity is high (Gemma: 0.980, Qwen: 0.968), but a fairer baseline is the cosine between *different* tools across languages (e.g., weather-English vs. search-Chinese). This cross-tool baseline is 0.926 for Gemma and 0.802 for Qwen. The gap, which measures how much of the similarity is actually about the tool rather than about the shared prompt, is 0.054 for Gemma and 0.167 for Qwen. The high baseline for Gemma means most of the shared activation comes from the common prompt context, not from tool-specific encoding; Qwen separates tools more cleanly across languages.

Classification on held-out queries. Using only English means to classify non-English queries, Qwen achieves 93%+ on all 8 languages. Gemma does well on European languages (French: 93%, Spanish: 87%) but drops on East Asian languages (Korean: 60%, Japanese: 67%). Adding tool descriptions to the prompt largely closes this gap (Korean: 60%→93%, Japanese: 67%→100%), suggesting that descriptions provide extra signal that helps bridge the cross-lingual gap.

A.24 Does adding tool descriptions change the results?

All steering experiments in the main text list tools as `name(params)`. Real-world function-calling systems usually also include a short description of what each tool does, like `weather(location): Check current weather conditions`. Here we test whether adding these descriptions changes the steering results.

The short answer: for IT models at 4B and above, name-only steering already works well (77–100%), and descriptions add at most a few percentage points. For smaller IT models the picture is mixed (descriptions help Gemma 1B but hurt Qwen 0.6B), and for base models descriptions consistently hurt, likely because the longer prompt overwhelms the model. Table 21 shows the full picture.

	IT		Base	
Model	Name	+Desc	Name	+Desc
Gemma 3				
270M	3%	3%	3%	3%
1B	57%	63% ↑	3%	3%
4B	97%	**100%** ↑	50%	20% ↓
12B	100%	100%	47%	37% ↓
27B	100%	100%	67%	63%
Qwen 3				
0.6B	67%	50% ↓	—	
1.7B	87%	87%	—	
4B	93%	**100%** ↑	—	
8B	100%	100%	—	
14B	100%	100%	—	
Others				
Qwen 2.5 7B	97%	97%	—	
Llama 3.1 8B	77%	83% ↑	77%	50% ↓

Table 21: Steering accuracy (exact match, 30 pairs) with and without tool descriptions. For IT models at 4B+, name-only steering already reaches 77–100%, and descriptions offer at most a small boost. Descriptions hurt the smallest IT model (Qwen 0.6B) and consistently hurt base models, where the longer prompt overwhelms the model.

The internal circuit does not change: attention heads still do 79–88% of the work with or without descriptions, and the same heads (like L17 H0 in Gemma) are the most important ones in both cases. Descriptions give the model more information to work with, but they do not change *how* the model processes it. For base models, descriptions can inflate the baseline through simple word over-

lap between the query and the tool prompt (e.g., "weather" appears in both the query and the description), bypassing the internal tool selection circuit entirely. This further supports our choice of name-only format for the primary evaluation.

A.25 How we test which components matter (activation patching)

To find out which parts of the network are actually responsible for picking the right tool, we use a simple swap test. We take a query that should trigger tool A (for example, "What is the weather in Tokyo?") and a different query that should trigger tool B ("Search for AI news."). We run both through the model and save what each component produces at each layer.

Then, one component at a time, we replace its output on the tool-A query with what it produced on the tool-B query, and check: does the model still pick tool A? If replacing a component causes the model to lose confidence in tool A, that component was important. If the model barely notices, the component was not doing much for tool selection.

We test every attention head and every feed-forward layer this way, across 10 different tool pairs, and average the results. On all three models we tested (Gemma 3 4B with 272 attention heads, Qwen 3 4B with 1,152 heads, and Llama 3.1 8B with 1,024 heads), the same pattern holds: a small fraction of heads (fewer than 5%) account for a disproportionate slice of the total effect, and the middle third of the network matters most (33–37%). Summed without normalization, attention totals 78–88% of the importance, but this is partly a counting artefact, since these models have many more attention heads than MLPs. Table 22 reports the same numbers on a per-component basis: individual MLPs are on average 1.94× as effective as individual heads in Gemma 3 4B and 4.19× in Qwen 3 4B. The takeaway is that a few specific heads (like L17 H0/H1 in Gemma) carry the decision; saying "attention dominates" is true in raw sum, but misleading on a per-component basis.

The right reading depends on the model. On Gemma 3 4B a few specific attention heads (L17 H0 and H1) really do dominate, with the top-3 heads carrying ∼2.7× the importance of the top-3 MLPs. On Qwen 3 4B that gap closes; top heads and top MLPs are tied, and the headline "88% attention" is almost entirely a side effect of having 32× more heads than MLPs.

A.26 Cosine readout method

For the BFCL readout comparison in Section 5, the procedure has two steps.

Step 1: pre-compute a mean-activation direction per tool. For each tool t, we take the 2 seed queries from the BFCL data for that tool, run the model forward through the full prompt (user query plus the BFCL-provided candidate list), and record the residual stream at the penultimate layer at the last token, which is the position where the model is about to emit the tool name. Averaging these gives a single direction $\bar{h}_t \in \mathbb{R}^d$ for the tool.

Step 2: read off the predicted tool on a new query. For a new query we do the same forward pass, get the residual at the last token, and compute its cosine similarity against every pre-computed tool direction. The argmax over tools is the readout prediction.

We use leave-one-out calibration across the BFCL samples per tool, strip any content inside `<think>...</think>` from generated text, and set `max_new_tokens = 200` for the generation baseline so that models with thinking modes (like Qwen 3) have room to emit a tool name after reasoning. An earlier draft used `max_new_tokens = 30`, which silently truncated thinking-mode outputs before any tool name was produced and inflated the gap between generation and readout; all numbers in Section 5 use the fair settings above.

BFCL v3 tool-name accuracy

Cosine geometry under matched prompt

A.27 Matched-prompt controls

K=15 matched-prompt control

For each of 15 synthetic tools we collect a tool-trigger query (e.g. "What is the weather in Tokyo" for `get_weather`). For the matched-prompt control we keep the same 15-tool prefix and same prompt template, but replace the user query with one of 15 non-tool topic queries (Roman history, quantum physics, economics, philosophy, literature, biology, music theory, geography, religion, mathematics, astronomy, anthropology, psychology, art history, linguistics; two queries per topic). We mean activations across the two queries, take the residual stream at the last token at the penultimate layer, and run PCA over the 15 means in each condition (Table 25).

Across all four models, non-tool topic queries actually use *fewer* dimensions than tool-trigger

	Total		Top-1		Top-3 sum		Top-5 sum	
	Attn	MLP	Attn	MLP	Attn	MLP	Attn	MLP
Gemma 4B (n_a=272, n_m=34)	73.3	17.8	6.48	2.13	12.6	4.7	16.1	6.8
Qwen 4B (n_a=1,152, n_m=36)	25.8	3.4	0.44	0.41	0.96	1.05	1.45	1.49

Table 22: Activation-patching importance for attention heads vs. MLP layers. Total: sum across all components. Top-k sum: sum of the k strongest individual contributors in each category. On Gemma 3 4B, top-k heads dominate top-k MLPs by 2–3× at every k. On Qwen 3 4B, top-k heads and top-k MLPs are essentially tied, so the 88% raw attention total is mostly a counting artefact (1,152 heads vs. 36 MLPs). Source: `results/circuit/per_component_normalization.json`, `results/circuit/topk_attn_vs_mlp.json`.

Model	Gen (%)	Readout (%)	Δ (pp)
Base models			
Gemma 3 1B base	2	69	+67
Gemma 3 4B base	3	75	+72
Gemma 3 12B base	∼5	61	+56
Llama 3.1 8B base	10	82	+72
Instruction-tuned models			
Gemma 3 4B IT	92	74	−17
Gemma 3 12B IT	95	66	−30
Llama 3.1 8B IT	90	53	−37

Table 23: BFCL v3 tool-name accuracy: greedy generation vs. cosine readout of the penultimate-layer residual against per-tool means (max_new_tokens=200, thinking tags stripped). Base models encode the tool before they can emit it; instruction tuning closes that output-side gap.

	Tool	Topic	Cross	Δ
Instruction-tuned				
Gemma 3 4B	0.960	0.957	0.916	+0.003
Gemma 3 12B	0.927	0.963	0.906	−0.036
Llama 3.1 8B	0.873	0.947	0.855	−0.074
Qwen 3 4B	0.792	0.945	0.779	−0.153
Base (pretrained-only)				
Gemma 3 4B-pt	0.999	0.999	0.998	−0.001
Gemma 3 12B-pt	0.998	0.997	0.996	+0.000
Llama 3.1 8B-base	0.922	0.944	0.892	−0.022

Table 24: Pairwise cosine of 200 mean activations under matched prompt prefix (last layer). Tool / Topic / Cross: cosines among tool means, among topic means, and between the two. Δ: tool-pair minus topic-pair cosine; negative means tool means are more spread. Source: `results/pca/cosine_analysis_summary.json`.

queries at this matched-prompt setup (k_{90} 7–9 vs. 10–11 for tools). The earlier "60% in 4 components for non-tool prompts vs. 100% for tools" framing in earlier drafts compared mismatched metrics (top-4 for non-tool against the trivial top-K-1 = 100% for tools); under a like-for-like K=15 comparison, the K=15 dimensionality reduction is not evidence of tool-specific compression. Tool means at K=15 fit

	Tools			Non-tool		
	t-4	t-10	k_{90}	t-4	t-10	k_{90}
Gemma 3 4B IT	0.61	0.92	10	0.71	0.94	9
Qwen 3 4B	0.57	0.91	10	0.77	0.96	8
Llama 3.1 8B	0.57	0.90	11	0.68	0.93	9
Gemma 3 12B IT	0.56	0.90	10	0.84	0.96	7

Table 25: K=15 PCA cumulative variance and k_{90} for tool-trigger queries vs. non-tool topic queries under the same 15-tool prompt prefix. t-4 / t-10: cumulative variance in the top 4 / top 10 components. Source: `results/pca/matched_control_k15_*.json`.

in ∼10 dimensions, but so do 15 distinct semantic topics under the same prompt prefix. The interventional evidence (steering, patching, within-topic probe) is what distinguishes tool identity, not PCA dimensionality.

K=200 matched-prompt control

At K=200 we run the same like-for-like control: 200 ToolBench API mean activations vs. 200 non-tool topic-query means under the same 200-tool prompt prefix.

The K=200 picture is model-by-model mixed (one similar, two spread, one cluster), so the random-Gaussian baseline alone does not isolate tool-specific compression at this scale; the cleaner evidence is the K=15 matched control above plus the within-topic probe and steering.

A.28 Anonymous-name control

To test how much of the tool subspace relies on the literal tool-name token, we replace the 15 tool names with anonymous IDs (`tool_00` through `tool_14`) in the prompt, keeping parameter lists and descriptions. Means, PCA, and steering are then recomputed against the anonymous names (Table 27).

The control addresses one confound (the "subspace is just the tool-name token" reading). It

	TOOLS	MATCH	DIFF	READING
Gemma 3 4B	36	39	+3	similar
Qwen 3 4B	80	52	−28	tools spread
Llama 3.1 8B	72	83	+11	tools cluster
Gemma 3 12B	84	11	−73	tools spread†

Table 26: k_{90} at K=200, last layer. TOOLS: 200 ToolBench API mean activations. MATCH: 200 non-tool topic queries with the same 200-tool prompt prefix. Negative DIFF means tool queries spread to more dimensions than non-tool queries; positive means they cluster more. †On Gemma 3 12B the non-tool baseline collapses (PC1=72%). Diff 95% CIs (subsample bootstrap at 80% of N, B=200): Gemma 3 4B [−5, +2] (crosses zero), Gemma 3 12B [+51, +56], Qwen 3 4B [+18, +26], Llama 3.1 8B [−17, −10]; so Gemma 3 4B's ∼0 difference and the other three signs are all statistically robust. Source: `results/pca/matched_control_summary.json`, `matched_control_subsample.json`.

	BASE		k_{90}		TOP-10	
	real	anon	real	anon	real	anon
Gemma 3 4B IT	100%	80%	10	9	0.91	0.93
Gemma 3 4B base	73%	47%	10	10	0.90	0.90
Qwen 3 4B	100%	100%	11	11	0.88	0.88
Llama 3.1 8B IT	100%	100%	11	11	0.89	0.87

Table 27: Anonymous-name control. BASE: tool-calling accuracy without any steering. TOP-10: cumulative variance of the top 10 PCA components. PCA geometry (k_{90} and top-10 variance) barely moves when the tool names are replaced by anonymous IDs, which rules out the reading that the low-dimensional structure is an artifact of tool-name tokens.

does not address the separate "subspace is just the description's topic" reading, which the within-topic analysis in Section 5 handles. Steering under anonymous IDs is a separate question we do not resolve here: the anonymous names share a common prefix (`tool_`), so a single-token or short-prefix match is uninformative, and any steering metric over them needs a finer-grained check against the full ID.

A.29 List-ordering control

This appendix backs up the list-ordering control in Section 4.1. We shuffle the 15-tool list into five orderings (identity, reverse, three random permutations with a fixed seed) and recompute each tool's mean activation in each ordering, using the same two queries per tool as the rest of Section 4.1. For each tool we then compare its five mean vectors pairwise by cosine similarity, recompute PCA k_{90} inside each ordering, and attribute variance between tool identity and list position (Table 28).

MODEL	COS	k_{90}	TOOL	POS	RATIO
Gemma 3 1B IT	0.94	9–10	16.3%	3.6%	4.5×
Gemma 3 4B IT	0.98	9–11	16.6%	3.9%	4.3×
Gemma 3 4B base	**1.00**	10–11	15.0%	3.3%	4.5×
Qwen 3 4B	0.96	11	17.6%	3.6%	4.9×
Llama 3.1 8B IT	0.96	11	17.0%	3.5%	4.9×

Table 28: List-ordering control. COS: mean pairwise cosine similarity of the same tool's mean activation across five orderings. k_{90}: the smallest number of PCA components to reach 90% variance (range across the five per-ordering runs). TOOL / POS: the fraction of total variance attributed to tool identity and to list position under an additive tool-plus-position decomposition. RATIO: tool variance divided by position variance. Across all five models, the same tool sits in essentially the same direction regardless of where it appears in the list, and tool identity accounts for roughly four to five times more variance than position.

The baseline tool-pick rate (no steering) under these orderings stays close to 100% on the instruction-tuned 4B+ models at every list position, with occasional dips at the last slot when a longer tool name happens to land there. On Gemma 3 1B IT, which has lower baseline accuracy overall, the dips are more scattered and not concentrated at any one position, which is consistent with a generally weaker tool-calling signal rather than a systematic primacy or recency bias.

A.30 Within-topic probe details

This appendix backs up the fifth piece of evidence against the topic-injection story in Section 5. Holding the topic roughly constant, we ask whether the linear signal can still tell tools apart.

τ-bench airline (same domain, different actions). All 14 airline tools share one topic but differ in action (book, cancel, update flights, update passengers, update baggages, search direct flight, search onestop flight, get user details, get reservation details, transfer to human agent, and so on). For each model, we take real multi-turn trajectories from the matched-context run, collect the residual stream at the position where the model is about to emit a tool name, and train a multinomial logistic-regression probe (80/20 stratified split, ℓ_2 regularization C=1, StandardScaler features). We only keep tools with at least 10 calls in the trajectory pool per model, so K varies between 5 and 9.

MODEL	K	CHANCE	TOP-1	TOP-5	RATIO
Gemma 3 4B IT	8	12.5%	61%	96%	4.9×
Gemma 3 12B IT	9	11.1%	66%	97%	5.9×
Qwen 3 4B	5	20.0%	71%	100%	3.5×
Qwen 3 8B	6	16.7%	72%	100%	4.4×
Qwen 3 14B	5	20.0%	89%	100%	4.5×

Table 29: Probe accuracy on τ-bench airline activations, restricted to tools called ≥ 10 times per model. Shuffle-label controls (labels permuted in training) are at 8–25%, consistent with chance. Random-Gaussian controls (features replaced by matched-shape noise) reach 18–42%; this is above chance because logistic regression can overfit noise at high d and small n, but the probe accuracy still sits 29–57 percentage points above this floor on every model. Claim: probe accuracy is well above random-data overfitting, not that specific numbers are tight under small-sample variance.

BFCL within-topic cluster (Movies). We run the same probe inside the Movies cluster of BFCL `live_multiple`, which contains three tools (`Movies_1_BuyMovieTickets`, `Movies_1_FindMovies`, `Movies_3_FindMovies`) that share the movies topic but do different things. With 10–20 real user queries per tool, the probe reaches 100% on all four models we tested (Gemma 3 4B / 12B, Qwen 3 4B, Llama 3.1 8B). The test set here is small (13 examples), so the noise-overfitting floor is higher than in the τ-bench airline setting (Gaussian baseline 38–46% vs chance 33%), leaving a 54–62 percentage point gap between the probe and that floor. Because the test set is this small we treat the Movies numbers as a consistency check rather than a tight standalone measurement; the τ-bench airline numbers in Table 29 are the primary test.

Cross-permutation control. The probe above uses real tool names that carry semantic content (`book_reservation`, `search_direct_flight`, and so on), so the linear signal could in principle be the model picking up the name string in the prompt. To test this we replay each trajectory under two anonymized permutations. Under **P1**, the 14 airline tools sorted alphabetically by real name map to fixed tags `act_A`–`act_N`; under **P2**, that mapping is cyclically shifted by seven, so every real tool is bound to a different tag than under P1. Renaming applies to the schema in the prompt and to every tool-call name in the dialog history; descriptions and parameter schemas stay verbatim. For every kept sample we capture the residual at the probe position under both permutations, then run four probes with a shared train/test split: within-P1, within-P2, P1→P2 (train on P1 activations, test on P2), and P2→P1.

MODEL	K	WP1	WP2	P1→P2	P2→P1
Gemma 3 4B IT	8	0.51	0.55	0.57	0.57
Gemma 3 12B IT	9	0.73	0.71	0.73	0.76
Qwen 3 4B	5	0.79	0.75	0.75	0.75
Qwen 3 8B	6	0.78	0.78	0.80	0.78
Qwen 3 14B	5	0.86	0.89	0.86	0.89

Table 30: Cross-permutation probe accuracy. WP1 and WP2 are within-permutation probes (train and test on the same permutation); P1→P2 and P2→P1 are transfer probes (train on one permutation, test on the other). Mean transfer accuracy exceeds mean within-permutation accuracy by 0.012 across the five models, and the gap is at most 0.04 in either direction on any single model, well within per-model binomial standard error (± 0.06–0.08, $n_{\text{test}} \in [24, 59]$).

Interpretation. A probe that was using topic alone should drop to chance once the topic is held constant. It does not. A probe whose direction was specific to a surface tag (e.g. "activate when `act_K` is being attended") should fail to transfer between permutations, because the same tag stands for different real tools under P1 and P2. It does not (Table 30). The linear direction the probe finds is therefore invariant to the surface tool-name string.

A.31 PCA scaling with ToolBench APIs

Table 31 shows the full PCA scaling curve using real API definitions from ToolBench (12,000+ APIs, stratified sampling across 49 domains, with descriptions).

K	k_{90}	MAX	RANDOM	COMPRESS
5	2	4	4.0	50%
10	5	9	8.7	56%
15	7	14	13.0	50%
20	8	19	17.0	42%
30	8	29	26.0	28%
50	17	49	43.0	35%
75	18	74	64.0	24%
100	26	99	86.0	26%
150	31	149	127.0	21%
200	36	199	167.0	18%
300	32	299	246.0	11%
500	57	499	392.0	11%

Table 31: PCA scaling on Gemma 3 4B with ToolBench APIs (stratified, with descriptions). At K=500 (10K tokens), k_{90}=57 vs. 392 for random, a 7× compression.

With vs. without descriptions. Without descriptions, the same model handles much larger tool

sets: k_{90} grows steadily from 3 (K=5) to 122 (K=2000) with no sign of collapse, because the prompt stays under 12K tokens even at K=2000. With descriptions, the model compresses better at moderate K (k_{90}=36 vs. 48 at K=200) but collapses beyond K=750 when the prompt exceeds 16K tokens. The practical recommendation: use descriptions for moderate tool sets ($K < 500$) where they improve compression, but switch to name-only format for very large sets to avoid the long-context penalty.

Cross-model comparison: attention heads predict subspace size. We repeat the PCA scaling on all 12 instruction-tuned models from three families. Table 32 shows the results.

MODEL	H	k_{90} AT K =					
		50	100	200	500	750	1000
Gemma 3 270M	4	17	27	50	46	69	74
Gemma 3 1B	4	20	33	48	41	96	83
Gemma 3 4B	8	17	26	36	57	27↓	22↓
Gemma 3 12B	16	28	50	84	129	130	—
Gemma 3 27B	16	29	47	62	131	108	130
Qwen 3 0.6B	16	24	46	67	138	151	164
Qwen 3 1.7B	16	25	49	77	133	144	139
Qwen 3 4B	32	26	49	80	151	154	168
Qwen 3 8B	32	24	47	80	147	158	153
Qwen 3 14B	40	29	55	89	171	171	184
Qwen 2.5 7B	28	29	55	89	173	179	178
Llama 3.1 8B	32	29	51	72	127	147	127↓

Table 32: PCA scaling across all 12 IT models (4 families). H = attention heads. Within Gemma 3, the 4B (8 heads) collapses at $K \geq 750$ but the larger 12B/27B (16 heads) recover. Sub-1B Gemma models (270M, 1B; 4 heads) are noisy with low absolute k_{90}. 16-32 head Qwen and Llama models keep growing or stay stable through K=1000. Qwen 3 4B reaches K=2000 with k_{90}=182; Qwen 2.5 7B and Qwen 3 1.7B reach K=1500 with k_{90}=279 and 230 respectively.

The pattern is clear: at K=50, sub-1B Gemma (4 heads) needs 17–20, 8-head Gemma 4B needs 17, 16-head models (Gemma 12B/27B, Qwen 0.6B/1.7B) need 24–29, and 28–40 head models (Qwen 8B/14B/2.5-7B, Llama 8B) need 24–29. At larger K, the gap widens dramatically: at K=500, Gemma 4B (8 heads) needs 57, Gemma 12B/27B (16 heads) need 129/131, Qwen 0.6B/1.7B (16 heads) reach 138/133, Llama 8B and Qwen 8B (32 heads) need 127/147, and the highest-head Qwen 14B (40 heads) needs 171. Within the Gemma family, scale + heads buys both compression and stability: 4B collapses past K=500, but 12B and 27B both stay near k_{90}=130 through K=1000, and 27B briefly dips at K=750 (108) before recovering. Sub-1B Gemma models (270M, 1B) are noisier and never reach the same absolute k_{90} as larger ones, but they avoid the catastrophic collapse pattern that Gemma 4B shows. Qwen also shows stronger cross-lingual tool alignment (Section A.23): its tool-specific gap is 3× larger (0.167 vs. 0.054) and it classifies Korean queries at 93% vs. Gemma's 60%. The extra dimensions may provide room for richer, more language-invariant tool representations, though we cannot rule out that the difference comes from pretraining data rather than architecture.

The 32+ head models also handle long tool lists best: at K=750 (16K tokens), Gemma 4B's representations collapse (k_{90} drops from 57 to 27), while Qwen 8B/14B/2.5-7B stay stable (147–179) and scale through K=1000. Qwen 4B keeps growing all the way to K=2000 (k_{90}=182). Larger Gemma variants (12B/27B) also recover their compression after the brief K=750 dip. Combining heads, d_{model}, and parameter count, the picture is: scale buys an effective context budget; once a model's combined capacity is exceeded, k_{90} collapses (Gemma 4B at K=750); larger models only delay this point.

Sampling sensitivity. We also tested a max-diversity sampling strategy (within each domain, prioritize APIs with different name prefixes). The curve shape changes (k_{90} at K=500: 30 with max-diversity vs. 57 with stratified), suggesting that the non-monotonic pattern reflects the real semantic structure of the API pool rather than a fixed property of the model.

Same-domain vs. cross-domain. When all tools come from a single domain (Finance, 500 APIs), the model needs only 32 dimensions at K=500, compared to 57 for the cross-domain stratified set. Within-domain tools are more similar to each other, so they pack into fewer directions. The Finance curve also shows an early plateau: k_{90} stays at 15 from K=50 to K=75, meaning the first 50 finance tools already span most of the "finance region" and adding more barely changes the subspace. This explains the non-monotonic dips in the main scaling curve: when a batch of new tools falls into a well-represented domain, k_{90} can stay flat or even dip.

Multi-domain comparison. We run the same analysis on individual domains to see how within-

domain similarity affects compression. Table 33 shows the results.

DOMAIN	5	15	50	100	200	500
Stratified (49)	2	7	17	26	36	57
Business	4	10	17	31	33	56
Finance	3	10	15	22	25	32
Data	3	8	16	26	22	39
Sports	3	7	14	24	33	11↓
Gaming	4	8	10	16	13	25
Entertainment	2	6	11	16	11↓	5↓
Weather†	3	7	21	21	—	—
Science†	3	8	14	—	—	—

Table 33: PCA k_{90} by domain across tool counts. At small K, all domains look similar (3–4 dimensions for 5 tools). At large K, domains diverge: diverse domains (Business, Stratified) keep growing, while homogeneous domains (Sports, Entertainment) plateau or *decrease*, because new tools fall into regions already covered. †Fewer APIs available.

Evaluation protocol

Steering evaluation. We report two metrics: *prefix match* (the model's first generated token matches the target tool's first token) and *exact match* (the first 5 generated tokens spell out the correct tool name). We use exact match as the primary metric throughout, except where noted. Each cell in our steering tables uses N=30 randomly sampled source→target pairs. We report one-sided binomial tests against the $1/K$ random baseline, with Wilson 95% confidence intervals (full CIs in Table 18). For IT-vs-base comparisons, we use Cohen's h as the effect size measure ($h \geq 0.5$ = medium, $h \geq 0.8$ = large). All p-values survive Bonferroni correction for the total number of tests.

BFCL evaluation. We use leave-one-out (LOO) cross-validation: for each test query, the tool means are computed from all *other* queries that map to the same function set, excluding the test example. This prevents data leakage. The generation parser (`parse_fixed`) handles Gemma's "`tool_code` output format by stripping the prefix, then tries exact-startswith matching followed by substring matching. Substring matching can inflate base model accuracy (e.g., if the model mentions "weather" in natural language text rather than as a structured tool call), so base model generation numbers should be interpreted as upper bounds.

PCA evaluation. We define k_{90} as the smallest number of principal components needed to capture 90% of the total variance across tool mean activations (formal definition in Appendix A.7). As a null baseline, we compute k_{90} for random points in $\mathbb{R}^d$ (Monte Carlo, 50–100 draws). As a specificity control, we also run a matched-prompt PCA where the 15-tool prefix is held fixed and only the user query content is swapped to a non-tool topic (Roman history, quantum physics, economics, etc.); see Appendix A.27 for results. Under this matched control the K=15 dimensionality is similar between tools and non-tool queries, so we do not lean on PCA dimensionality alone for tool-specific claims.