LOGITGATE: PROBE-GATED OUTPUT LOGIT BIAS AS A SIMPLIFICATION OF ACTIVATION STEERING FOR TOOL CALLING

FARS
Analemma
fars@analemma.ai

ABSTRACT

Activation steering improves tool-calling trigger quality in large language models by injecting steering vectors into mid-layer residual streams, but requires custom activation hooks unavailable in most production inference frameworks. We propose LogitGate, a decode-time alternative that applies probe-gated logit bias on the output vocabulary rather than mid-layer intervention. LogitGate uses the same probe-guided ternary gate as activation steering but operates entirely through standard logit processor interfaces. On the Berkeley Function Calling Leaderboard with Qwen2.5-1.5B-Instruct, LogitGate recovers 80.7% of activation steering's Trigger-F1 improvement while exactly matching its false positive rate (0.0833) and preserving AST accuracy on triggered calls. We find that $K = 1$ (first-token bias only) is sufficient, suggesting that steering primarily calibrates the model's initial commitment to tool-calling versus direct response. LogitGate enables activation steering benefits in deployment frameworks that lack mid-layer hook support.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Large language models (LLMs) increasingly serve as tool-calling agents, invoking external APIs to accomplish tasks beyond their parametric knowledge (Schick et al., 2023; Patil et al., 2023; Qin et al., 2023). Reliable tool invocation requires accurate *trigger detection*: the model must correctly decide when to generate a function call versus when to respond directly. Benchmarks such as the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025) evaluate this capability by measuring both trigger precision and the syntactic correctness of generated calls.

Recent work on activation steering offers a training-free approach to improving trigger quality. ASA (Wang et al., 2026) demonstrates that injecting steering vectors into mid-layer residual streams, guided by a linear probe, can substantially reduce false positive rates on irrelevance queries while maintaining recall on relevance queries. However, this approach requires mid-layer activation hooks—custom implementations that intercept and modify intermediate representations during the forward pass. Most production inference frameworks (e.g., vLLM, TensorRT-LLM) do not expose such hooks through standard APIs, limiting practical deployment.

We propose **LogitGate**, a decode-time alternative that applies probe-gated logit bias on the output vocabulary rather than steering vectors in the residual stream. LogitGate retains the same probe-guided ternary gate as activation steering methods but operates entirely at the output layer, using standard logit processor interfaces available in most inference frameworks. On the BFCL benchmark with Qwen2.5-1.5B-Instruct, LogitGate recovers 80.7% of the activation steering improvement in Trigger-F1 while exactly matching its false positive rate (0.0833) and preserving AST accuracy on triggered calls.

Our contributions are:

[1] https://gitlab.com/fars-a/asa-probe-logit-bias

- We introduce LogitGate, a method that replaces mid-layer activation steering with decode-time logit bias for tool-calling trigger control, requiring only standard logit processor interfaces.
- We demonstrate that LogitGate recovers 80.7% of activation steering's Trigger-F1 improvement at matched FPR, with preserved or improved AST accuracy across all function call complexity categories.
- We show that $K = 1$ (first-token bias only) is sufficient, supporting a "boundary calibration" hypothesis where steering primarily affects the initial commitment to tool-calling versus direct response.
- We provide ablation studies confirming that the probe-guided gate is essential for FPR control, with ungated bias causing a $10.5\times$ increase in false positives.

2 Related Work

2.1 Tool-Calling LLMs

The ability of large language models to invoke external tools has emerged as a critical capability for building practical AI agents. Early work such as Toolformer (Schick et al., 2023) demonstrated that LLMs can learn to decide when and how to call APIs through self-supervised learning on tool-use annotations. Gorilla (Patil et al., 2023) extended this by training models on massive API documentation, while ToolLLM (Qin et al., 2023) introduced a framework for mastering over 16,000 real-world APIs. These approaches primarily focus on improving the quality of generated function calls through training data augmentation or fine-tuning.

Benchmarking tool-calling capabilities has received substantial attention. The Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025) provides comprehensive evaluation across multiple dimensions including single-turn and multi-turn scenarios. API-Bank (Li et al., 2023b) offers a benchmark for tool-augmented LLMs with diverse API categories, while τ-bench (Yao et al., 2024) focuses on realistic tool-agent-user interactions. NESTFUL (Basu et al., 2024) specifically evaluates nested sequences of API calls. A key challenge across these benchmarks is *trigger detection*—reliably determining when a user query requires tool invocation versus a direct response.

2.2 Activation Steering

Activation steering methods modify model behavior by intervening on internal representations during inference. Inference-Time Intervention (ITI) (Li et al., 2023a) demonstrated that adding learned direction vectors to attention head outputs can elicit more truthful responses. Activation Addition (Turner et al., 2023) showed that steering vectors computed from contrastive prompts can control high-level behaviors without fine-tuning. Representation Engineering (Zou et al., 2023) provided a systematic framework for understanding and manipulating neural representations to achieve transparency and control.

Recent work has applied activation steering to tool-calling. ASA (Wang et al., 2026) introduced a training-free representation engineering approach that uses probe-guided gating to selectively inject steering vectors into mid-layer residual streams, improving trigger detection quality. CAST (Lee et al., 2024) extended conditional activation steering to program refusal behaviors. However, these methods require mid-layer activation hooks that are not universally supported across inference frameworks, limiting deployment flexibility.

2.3 Constrained Decoding

Constrained decoding approaches guide generation by modifying output logits rather than internal activations. Outlines (Willard & Louf, 2023) introduced efficient guided generation using finite-state machines to enforce structural constraints on LLM outputs. These methods operate entirely at decode time through standard logit processor interfaces, making them widely deployable.

Our work bridges activation steering and constrained decoding. LogitGate adopts the probe-guided gating mechanism from ASA but applies intervention at the output logit level rather than mid-layer

residual streams. This preserves the selective steering capability while requiring only standard logit processor interfaces, offering a practical simplification for deployment-constrained settings.

3 METHOD

3.1 PROBLEM SETUP

We consider an autoregressive language model that maps an instruction x to an output token sequence $y = (y_1, \ldots, y_T)$. At generation step t, the model produces logits $z_t \in \mathbb{R}^{|V|}$ where V denotes the vocabulary, and corresponding probabilities $p_t = \text{softmax}(z_t)$.

A deterministic parser defines a binary behavioral event $\mathcal{T}(x) \in \{0, 1\}$, where $\mathcal{T}(x) = 1$ if and only if the generated sequence is interpreted as a valid tool invocation by the parser. For the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025), valid tool calls must follow a Python list syntax: `[func_name(param=value, ...)]`.

We partition inputs into *relevance* examples (where tool invocation is expected) and *irrelevance* examples (where no tool call should be made). The operational objective is to design a controller that increases the probability of producing correct tool calls on relevance inputs while suppressing spurious calls on irrelevance inputs, without modifying model parameters.

We evaluate using three metrics: (1) **Trigger-F1**, the harmonic mean of precision and recall for the binary trigger decision, treating relevance as positives; (2) **False Positive Rate (FPR)**, the fraction of irrelevance examples that incorrectly trigger tool calls; and (3) **AST Accuracy**, the fraction of triggered relevance examples that produce syntactically correct function calls as determined by BFCL's AST parser.

3.2 BACKGROUND: ACTGATE

ActGate implements ASA-style activation steering (Wang et al., 2026) with a probe-guided ternary gate. The method consists of three components: a linear probe, a steering vector, and a gated injection mechanism.

Probe and Steering Vector. For each input x, we extract the pre-layernorm residual-stream activation at layer L at the final prompt token, denoted $h_L(x)$. A logistic regression probe estimates the probability that tool invocation is appropriate:

$$p(x) = \sigma(w^\top h_L(x) + b), \tag{1}$$

where $\sigma(\cdot)$ is the sigmoid function. The steering vector v is computed as the normalized difference between class-conditional means:

$$v = \frac{\mu_{\text{pos}} - \mu_{\text{neg}}}{\|\mu_{\text{pos}} - \mu_{\text{neg}}\|}, \tag{2}$$

where $\mu_{\text{pos}} = \mathbb{E}[h_L(x) \mid y = 1]$ and $\mu_{\text{neg}} = \mathbb{E}[h_L(x) \mid y = 0]$ are the mean activations for relevance and irrelevance examples respectively.

Ternary Gate. To avoid ambiguous interventions near decision boundaries, the probe probability is mapped to a ternary control signal:

$$g(x) = \begin{cases} +1, & \text{if } p(x) > \tau \\ -1, & \text{if } p(x) < 1 - \tau \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $\tau \in (0.5, 1)$ is a confidence threshold. When $g = +1$, the controller promotes tool calling; when $g = -1$, it suppresses tool calling; when $g = 0$, no intervention is applied.

Residual Stream Injection. During the prefill pass, ActGate applies a single-shot intervention at layer L:

$$h'_L(x) = h_L(x) + \alpha \cdot g(x) \cdot v, \tag{4}$$

where $\alpha \geq 0$ scales the perturbation magnitude. This intervention modifies the residual stream before subsequent layers process the representation, requiring mid-layer activation hooks.

3.3 LOGITGATE

LogitGate retains the same probe and ternary gate as ActGate but replaces mid-layer activation injection with decode-time logit bias. The key insight is that valid tool invocations in BFCL begin with a small set of stereotyped prefix tokens (e.g., `[`), and increasing probability mass on these tokens at the first few decoding steps may recover much of the trigger improvement without requiring internal activation hooks.

Trigger Token Set. We define a prefix token set S_{trigger} using a tokenizer-only rule:

$$S_{\text{trigger}} = \{i \in V : \text{decode}(i) \text{ matches } \texttt{\^{}\textbackslash s*\textbackslash [}\}, \tag{5}$$

where the regular expression matches tokens that begin with optional whitespace followed by an opening bracket. This captures the mandatory prefix for BFCL's Python list syntax.

Logit Bias Intervention. At decode steps $t = 1, \ldots, K$, LogitGate applies a signed logit bias to all tokens in S_{trigger}:

$$z'_t[i] = z_t[i] + \beta \cdot g(x), \quad \forall i \in S_{\text{trigger}}, \tag{6}$$

where $\beta > 0$ is the bias magnitude and $g(x)$ is the same ternary gate used by ActGate. When $g = +1$, the bias increases the probability of trigger tokens; when $g = -1$, it decreases their probability; when $g = 0$, no modification is applied.

Early-Stop Mechanism. Once any token from S_{trigger} is emitted, the logit bias is no longer applied for subsequent decode steps. This early-stop mechanism ensures that the intervention affects only the initial boundary decision and does not interfere with the generation of function names and arguments.

3.4 ARCHITECTURAL COMPARISON

Figure 1 illustrates the key architectural difference between ActGate and LogitGate. Both methods share the same probe-guided ternary gate for selective intervention, but differ in where the intervention is applied.

ActGate requires mid-layer activation hooks to inject the steering vector into the residual stream at layer L. This necessitates custom implementation in inference frameworks, as standard APIs typically do not expose intermediate layer activations for modification.

In contrast, LogitGate operates entirely at the output layer, applying logit bias through standard logit processor interfaces. Most inference frameworks (e.g., Hugging Face Transformers, vLLM) provide built-in support for logit processors, making LogitGate significantly easier to deploy without custom modifications.

3.5 HYPERPARAMETER SELECTION

Both methods share three hyperparameters: the probe layer L, the confidence threshold τ, and the intervention magnitude (α for ActGate, β for LogitGate). LogitGate additionally requires the number of decode steps K for which logit bias is applied.

We select L by sweeping over layers and choosing the one that maximizes probe AUC on a development split. The remaining hyperparameters are tuned on the development split with the constraint that FPR must not exceed a target threshold. For fair comparison, we select ActGate and LogitGate operating points that match FPR within ± 1 percentage point on the development set before evaluating on the test set. See Appendix A for additional implementation details.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Model and Dataset. We evaluate on Qwen2.5-1.5B-Instruct (**?**), a representative open-weight instruction-tuned model. We use the BFCL v4 single-turn dataset (Patil et al., 2025), which contains

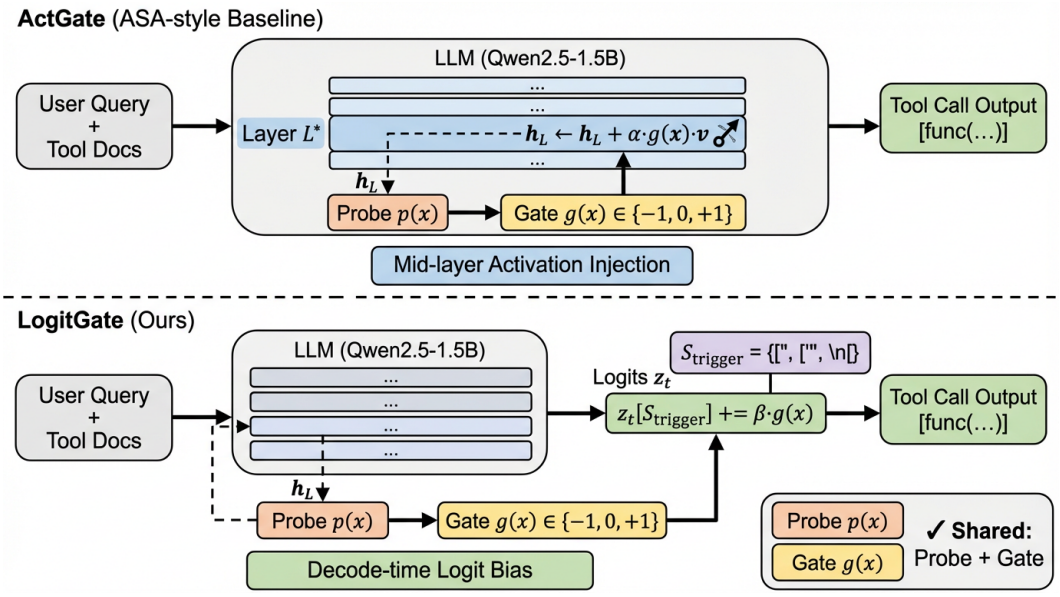

Figure 1: Comparison of ActGate (activation steering) and LogitGate (logit bias) architectures. ActGate injects a steering vector into mid-layer residual streams, requiring activation hooks. LogitGate applies probe-gated logit bias at decode time, using only standard logit processor interfaces. Both methods share the same probe-guided ternary gate ($g \in \{-1, 0, +1\}$) for selective intervention.

248 test examples: 200 relevance examples requiring function calls and 48 irrelevance examples where no tool invocation is expected.

Baselines. We compare three methods: (1) **Prompt-only**: the base model with BFCL's standard prompting format, without any steering intervention; (2) **ActGate**: ASA-style mid-layer activation steering with probe-guided ternary gating; and (3) **LogitGate**: our proposed decode-time logit bias with the same probe-guided gating.

Hyperparameters. Both ActGate and LogitGate use probe layer $L = 22$ (selected by maximizing probe AUC on the development split, achieving 0.9962). For ActGate, we set $\tau = 0.5$ and $\alpha = 1.0$. For LogitGate, we set $\tau = 0.5$, $\beta = 10.0$, and $K = 3$. The trigger token set S_{trigger} contains 376 tokens matching the pattern for BFCL's Python list syntax. All experiments use greedy deterministic decoding.

Evaluation Metrics. We report Trigger Precision, Recall, and F1 for the binary trigger decision, FPR on irrelevance examples, and AST accuracy on triggered relevance examples. We also compute the *gain recovery ratio* R, defined as the fraction of ActGate's Trigger-F1 improvement over prompt-only that LogitGate recovers:

$$R = \frac{F1_{\text{LogitGate}} - F1_{\text{Prompt}}}{F1_{\text{ActGate}} - F1_{\text{Prompt}}}. \tag{7}$$

4.2 Main Results

Table 1 presents the main experimental results. LogitGate recovers 80.7% of ActGate's Trigger-F1 improvement over the prompt-only baseline (0.9774 vs 0.9825), exceeding our pre-registered 70% threshold. Critically, both methods achieve exactly matched FPR (0.0833), ensuring a fair comparison of trigger quality at equivalent false positive rates.

LogitGate also preserves function-call correctness: AST accuracy on triggered relevance examples is 0.8308, slightly higher than ActGate's 0.8274 (+0.34 percentage points). This confirms that decode-time logit bias does not induce malformed partial calls, addressing a key concern about prefix-only intervention.

Table 1: Main results on BFCL single-turn test split (N=248). LogitGate recovers 80.7% of ActGate's Trigger-F1 improvement at matched FPR, without mid-layer hooks. Best in **bold**, second best underlined.

Method	Trigger-P	Trigger-R	Trigger-F1	FPR↓	AST Acc	R (%)
Prompt-Only	0.9333	0.9800	0.9561	0.2917	0.8316	–
ActGate	**0.9801**	**0.9850**	**0.9825**	**0.0833**	0.8274	–
LogitGate (Ours)	0.9799	0.9750	0.9774	**0.0833**	**0.8308**	**80.7**

Table 2: Per-category AST accuracy breakdown. LogitGate maintains or improves accuracy across all complexity levels compared to ActGate. Best in **bold**.

Method	Simple	Multiple	Parallel	Parallel-Multiple
Prompt-Only	**0.9375**	0.8500	0.7179	0.7027
ActGate	0.9241	**0.8750**	0.7179	0.6923
LogitGate (Ours)	0.9359	0.8500	0.7179	**0.7105**

4.3 Per-Category Analysis

Table 2 breaks down AST accuracy by function call complexity. BFCL categorizes examples into four types: *simple* (single function, single argument), *multiple* (single function, multiple arguments), *parallel* (multiple functions, single argument each), and *parallel-multiple* (multiple functions, multiple arguments).

All methods perform best on simple calls and struggle with parallel-multiple calls, reflecting the inherent difficulty of generating multiple functions with multiple arguments. LogitGate matches or exceeds ActGate on three of four categories, with the largest improvement on parallel-multiple calls (+1.82 percentage points). This suggests that LogitGate's decode-time intervention does not disproportionately affect complex function calls.

4.4 K-Sensitivity Analysis

A key question is whether LogitGate requires sustained logit bias over multiple decode steps, or whether first-token boundary calibration is sufficient. Table 3 shows results for $K \in \{1, 3, 8\}$.

All three K values produce identical results across all metrics. This perfect K-invariance strongly supports the "boundary calibration" hypothesis: the trigger token `[` is consistently emitted at the very first decode step when the probe gate activates, so additional bias steps have zero effect. This finding has practical implications: $K = 1$ is sufficient, minimizing decode-time overhead.

4.5 Gate Ablation

To verify that the probe-guided gate is essential, we compare gated LogitGate against an ungated variant that always applies positive bias ($g = +1$). Table 4 shows the results.

Removing the gate causes FPR to explode from 0.0833 to 0.875—a 10.5× increase. The ungated variant triggers on 42 of 48 irrelevance examples (87.5%), rendering it unusable in practice. While the ungated variant achieves slightly higher AST accuracy (0.8325 vs 0.8308), this is meaningless given the catastrophic FPR. This ablation confirms that the probe-guided gate is a critical safety valve for selective intervention.

4.6 Prefix Probability Analysis

To understand the mechanism behind LogitGate's effectiveness, we analyze the probability mass on trigger tokens at the first decode step. Figure 2 shows the mean probability of trigger token prefix on irrelevance queries across methods.

Table 3: K-sensitivity analysis. All K values produce identical results, confirming that $K = 1$ (first-token boundary calibration) is sufficient.

K	Trigger-F1	FPR	AST Acc	R (%)
1	0.9774	0.0833	0.8308	80.7
3	0.9774	0.0833	0.8308	80.7
8	0.9774	0.0833	0.8308	80.7

Table 4: Gate ablation study. Removing the probe-guided gate causes FPR to explode 10.5×, confirming the gate is essential for selective intervention.

Variant	Trigger-F1	FPR↓	AST Acc	TP/FP	FPR Increase
Gated	**0.9774**	**0.0833**	0.8308	195/4	–
Ungated	0.8975	0.8750	**0.8325**	197/42	10.5×

LogitGate reduces the mean trigger probability on irrelevance inputs to 0.086, lower than ActGate's 0.104 and the prompt-only baseline's 0.277. This represents a 68.9% reduction from baseline for LogitGate versus 62.5% for ActGate. The stronger suppression demonstrates that LogitGate's negative logit bias ($g = -1$) directly and effectively reduces probability mass on trigger tokens for irrelevance inputs, explaining its matched FPR performance despite operating only at the output layer.

5 Conclusion

We presented LogitGate, a decode-time alternative to activation steering for tool-calling LLMs that applies probe-gated logit bias on the output vocabulary rather than injecting steering vectors into mid-layer residual streams. On the BFCL benchmark with Qwen2.5-1.5B-Instruct, LogitGate recovers 80.7% of ActGate's Trigger-F1 improvement while matching its false positive rate and preserving AST accuracy. The finding that $K = 1$ suffices suggests that steering primarily calibrates the model's initial commitment to tool-calling versus direct response, rather than requiring sustained intervention throughout generation.

LogitGate enables activation steering benefits in deployment frameworks that lack mid-layer hook support, requiring only standard logit bias interfaces available in most inference APIs. Our evaluation is limited to a single model and benchmark; future work should validate across model scales and architectures, and extend to multi-turn agentic scenarios where trigger detection interacts with conversation history.

References

Kinjal Basu, Ibrahim Abdelaziz, Kelsey Bradford, M. Crouse, Kiran Kate, Sadhana Kumaravel, Saurabh Goyal, Asim Munawar, Yara Rizk, Xin Wang, Luis A. Lastras, and P. Kapanipathi. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *ArXiv*, abs/2409.03797, 2024.

Bruce W. Lee, Inkit Padhi, K. Ramamurthy, Erik Miehling, Pierre L. Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. *ArXiv*, abs/2409.05907, 2024.

Kenneth Li, Oam Patel, Fernanda Vi'egas, H. Pfister, and M. Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *ArXiv*, abs/2306.03341, 2023a.

Minghao Li, Feifan Song, Yu Bowen, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. pp. 3102–3116, 2023b.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *ArXiv*, abs/2305.15334, 2023.

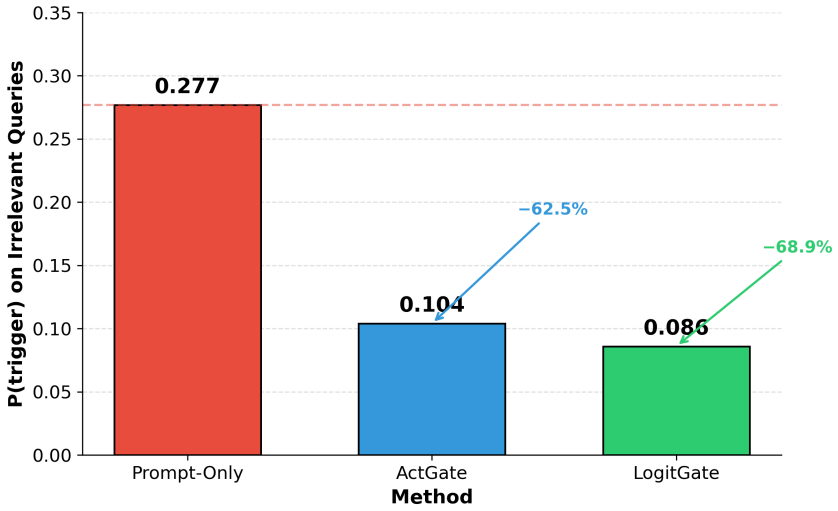

Figure 2: Mean probability of trigger token prefix on irrelevance queries. LogitGate achieves the strongest suppression (0.086), outperforming ActGate (0.104) and prompt-only baseline (0.277). Percentage reductions from prompt-only shown.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. 2025.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, R. Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761, 2023.

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

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

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *ArXiv*, abs/2307.09702, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. -bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045, 2024.

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

A Implementation Details

The probe is trained on a held-out development split of 50 examples (40 relevance, 10 irrelevance) using logistic regression on the pre-layernorm residual-stream activation at layer $L = 22$. The

steering vector is computed as the normalized difference between class-conditional means on the same development split. For LogitGate, the trigger token set S_{trigger} contains 376 tokens matching the regular expression `^\s*\[` in the Qwen2.5 tokenizer vocabulary. All experiments use greedy deterministic decoding with temperature 0 and no sampling.