TOOLSAE: DATA SELECTION FOR TOOL LEARNING VIA CAPABILITY-RELEVANT NEURON ACTIVATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool calling is a foundational capability that enables large language model (LLM) agents to interact with the external world. Training such capabilities increasingly relies on large-scale tool-calling traces, making it critical to identify the most useful traces from these extensive datasets. Existing data selection methods typically prioritize alignment with target examples or generic criteria such as quality and diversity, yet these signals often capture only surface features (e.g., schema keywords), while failing to reflect the specific tool-calling capabilities to be strengthened. In this work, we introduce TOOLSAE, a capability-guided data selection framework that explicitly identifies tool-capability neurons in the SAE latent space and selects training examples based on their activations. We employ sparse autoencoders to decompose polysemantic model activations into sparse, interpretable representations. Through contrastive neuron profiling and token-level diagnosis, TOOLSAE identifies capability-specific neurons while filtering out those responding primarily to surface formatting. It then ranks candidate traces by their weighted activations over the retained neurons, prioritizing examples that engage tool-calling behaviors. Empirical evaluations show that training on just 5% of the tool-calling data selected by TOOLSAE achieves tool-calling performance competitive with full-data training while maintaining comparable general-domain performance. Further analysis shows that the identified neurons capture tool-calling semantics and provide effective signals for capability-guided data selection.

1 INTRODUCTION

Large language models (LLMs) demonstrate strong language, reasoning, and coding capabilities, highlighting their potential as intelligent agents. Realizing this potential requires interaction with the external world to retrieve information, execute computations, and take actions. Tool calling provides this interface by translating user intent into executable operations (Schick et al., 2023; Qin et al., 2024). This interest has driven the development of many tool-augmented models and agent systems (Zhang et al., 2025). A central challenge is how to effectively train tool-calling capabilities across diverse tools and interaction patterns.

Recent synthesis pipelines expand tool-calling datasets in scale, API diversity, and interaction complexity. APIGen combines automated generation with execution and semantic verification to produce reliable traces across diverse APIs (Liu et al., 2024b). ToolACE synthesizes complex dialogues through multi-agent interactions and applies rule- and model-based checks to improve data quality (Liu et al., 2025). However, a larger candidate pool does not always lead to better tool-calling performance. Redundant or low-quality examples can make full-data training less effective than training on a carefully selected subset (Li et al., 2024b). We therefore ask (§ 2.1): *Is it necessary to expose the model to every candidate trace to effectively build up tool-calling capabilities?*

Data selection has become an important approach to improving the data efficiency of LLM training. Existing methods select for alignment with target examples (e.g., BM25 and DSIR) (Robertson & Zaragoza, 2009; Xie et al., 2023), instruction-following difficulty (Li et al., 2024b), or estimated influence on target performance (e.g., LESS and MATES) (Xia et al., 2024; Yu et al., 2024). Applying these criteria to tool calling presents two limitations. First, tool calling is an abstract capability spanning tool selection, argument reasoning, and multi-step invocation (Figure 1-a). A small set of target

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

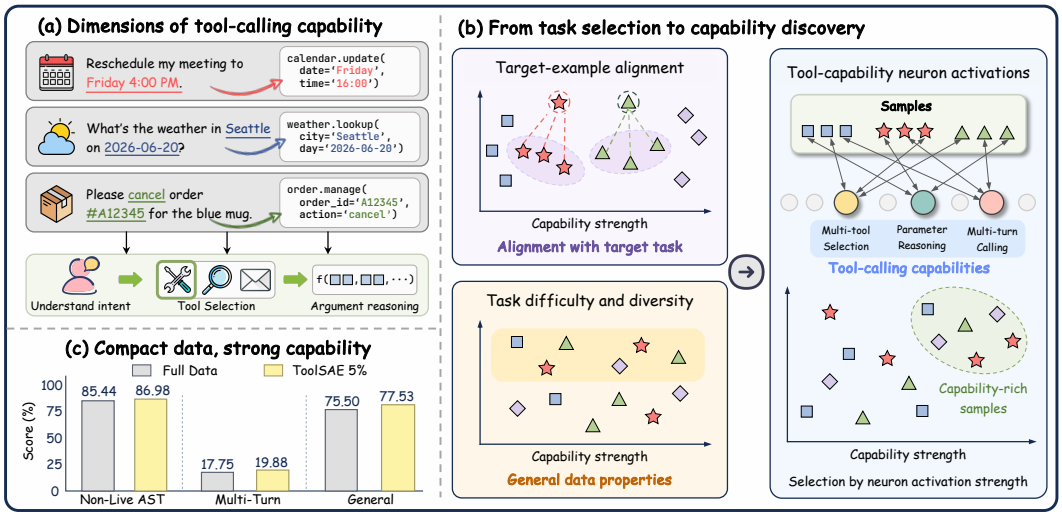

Figure 1: Overview of TOOLSAE. **(a)** Dimensions of tool-calling capability. **(b)** Existing methods prioritize target alignment or generic data properties, while TOOLSAE selects examples based on capability-specific neuron activations. **(c)** Performance comparison between training on TOOLSAE-selected data and full-data training.

examples is unlikely to capture all these dimensions, limiting the coverage of target-guided selection. Second, data selection can favor surface formatting, such as schema keywords and JSON structure, without reflecting how strongly an example engages tool-calling capabilities. Even activation-based methods emphasize target alignment or feature diversity (Ma et al., 2026; Yang et al., 2025), without explicitly isolating capability-specific responses.

Our motivation draws on neuroscience studies showing that learning a physical skill is associated with consistent activity patterns in neurons involved in that skill (Peters et al., 2014). Inspired by this link between skills and neural activity, we propose selecting training examples based on how strongly they engage the model's tool-calling capabilities. Specifically, we hypothesize that examples strongly activating tool-capability neurons provide useful signals for capability learning. This requires addressing two challenges: 1) *how to capture tool-calling capabilities through model-internal activation patterns*, and 2) *how to reliably identify the relevant neurons*.

We propose TOOLSAE, a capability-guided data selection framework that selects training examples based on capability-specific neuron activations (Figure 1-b; § 2). First, we use sparse autoencoders (SAEs) (Huben et al., 2024; Gao et al., 2025) to disentangle polysemantic model activations into sparse, monosemantic units, which we refer to as neurons in the SAE latent space. Second, *contrastive neuron profiling* identifies SAE neurons associated with tool-calling behaviors, which we refer to as tool-capability neurons. It contrasts tool and non-tool traces, measures specificity to different tool-calling behaviors, and penalizes associations with formatting and data sources. Token-level diagnosis further filters out shortcut neurons whose strongest responses concentrate on surface formatting. Finally, TOOLSAE ranks candidate traces by their weighted activations over the retained neurons and selects the top-k examples for training.

Empirical evaluations show that training on just 5% of the tool-calling data selected by TOOLSAE outperforms full-data training on Non-Live AST and Multi-Turn evaluations while maintaining comparable general-domain performance (Figure 1-c). Further analysis shows that the identified neurons capture tool-calling semantics beyond surface formatting, and ablations support the effectiveness of neuron identification and diagnosis. In summary, our contributions are:

- We propose TOOLSAE, a capability-guided data selection framework that explicitly identifies tool-capability neurons and selects training examples based on their activations.
- We develop contrastive neuron profiling and token-level diagnosis to identify capability-specific neurons and filter out formatting shortcuts.

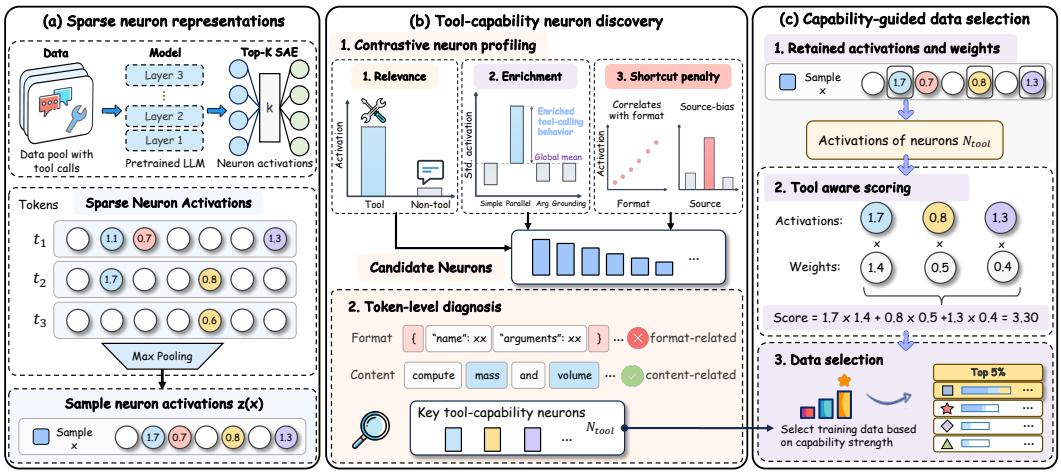

Figure 2: Overview of TOOLSAE. **(a)** A Top-K SAE encodes token-level hidden states into sparse activations, which are max-pooled into a sample representation. **(b)** Contrastive neuron profiling combines relevance, enrichment, and a shortcut penalty; token-level diagnosis removes neurons dominated by formatting tokens to obtain $\mathcal{N}_{\text{tool}}$. **(c)** Weighted activations over the retained neurons score pool examples, and the highest-scoring examples form the selected training subset (top 5% in our experiments). Activation values and weights are illustrative.

- We validate TOOLSAE through empirical experiments and provide ablation and interpretability analyses of the identified neurons.

2 METHODOLOGY

We introduce TOOLSAE, a capability-guided data selection framework that identifies tool-capability neurons and uses their activations to prioritize training examples (Figure 2). In this section, we (i) formalize the tool-calling data selection problem and its objective, (ii) construct sparse neuron representations of candidate examples, (iii) identify tool-capability neurons through contrastive neuron profiling and token-level diagnosis, and (iv) select training examples based on their weighted activations.

2.1 PROBLEM FORMULATION

Given a base model $\mathcal{M}_\theta$ and a candidate pool of tool-calling conversations $\mathcal{D}^{\text{pool}} = \{x_i\}_{i=1}^{N}$, our goal is to select a subset of k examples that best improves tool-calling performance. Here, k is a fixed data budget. For any candidate subset $\mathcal{D} \subseteq \mathcal{D}^{\text{pool}}$, the trained model is obtained by supervised fine-tuning (SFT) on $\mathcal{D}$ together with a fixed set of general instructions $\mathcal{D}^{\text{gen}}$:

$$\mathcal{M}_{\theta^*(\mathcal{D})} = \text{SFT}(\mathcal{M}_\theta, \mathcal{D} \cup \mathcal{D}^{\text{gen}}). \tag{1}$$

Including general instructions reflects a practical tuning setting in which models are expected to handle general requests as well as tool calls. Let $\mathcal{T}^{\text{tool}}$ denote the tool-calling evaluation tasks and $\mathcal{E}(\mathcal{M}, \mathcal{T})$ the performance of model $\mathcal{M}$ on tasks $\mathcal{T}$. The data selection objective is

$$\mathcal{D}^{\text{sel}} = \underset{\mathcal{D} \subseteq \mathcal{D}^{\text{pool}},\, |\mathcal{D}|=k}{\arg\max} \; \mathcal{E}\big(\mathcal{M}_{\theta^*(\mathcal{D})}, \mathcal{T}^{\text{tool}}\big). \tag{2}$$

The general-instruction set and SFT configuration remain fixed across data selection methods, so the comparison isolates the effect of the selected tool-calling data.

2.2 SPARSE NEURON REPRESENTATIONS

To identify tool-capability neurons, we first need a representation that separates the semantic concepts mixed within model activations. Raw hidden dimensions are polysemantic: the same dimension can respond to several unrelated concepts, making it difficult to determine which responses

reflect tool-calling capabilities. Sparse autoencoders (SAEs) encourage sparse, monosemantic representations in which individual latent neurons capture more distinct semantic concepts (Huben et al., 2024; Gao et al., 2025). This separation makes neuron responses easier to interpret and provides a basis for identifying capability-specific neurons in the next step. As shown in Figure 2-a, we encode the hidden state $\mathbf{h}_p \in \mathbb{R}^d$ of token p with a Top-K SAE:

$$\mathbf{z}_p = \mathrm{TopK}(\mathbf{W}_{\mathrm{enc}}(\mathbf{h}_p - \mathbf{b}_{\mathrm{dec}})), \qquad \mathbf{z}_p \in \mathbb{R}^{d'}. \tag{3}$$

Each dimension of $\mathbf{z}_p$ represents the activation of a neuron in the SAE latent space. Evidence of tool-calling behavior may appear in only a few tokens, such as a function name or an argument value drawn from the conversation. We therefore aggregate token activations with dimension-wise max pooling,

$$\mathbf{z}(x)[j] = \max_{p \in \{1,\ldots,|x|\}} \mathbf{z}_p[j], \tag{4}$$

which preserves each neuron's strongest response within the example.

2.3 Tool-Capability Neuron Identification

Using the sample representations $\mathbf{z}(x)$, we identify tool-capability neurons through contrastive neuron profiling and token-level diagnosis (Figure 2-b). Profiling scores neurons by their relevance to tool calling, specificity to different behaviors, and associations with formatting and data sources. Diagnosis then inspects their token-level responses to filter out formatting shortcuts.

Contrastive neuron profiling. The first step in Figure 2-b evaluates candidate neurons at the sample level. We contrast a set of tool traces $\mathcal{D}^{\mathrm{tool}}$ with non-tool instructions $\mathcal{D}^{\mathrm{nontool}}$. A useful neuron should satisfy three criteria. First, it should be more active on tool traces. We measure this *relevance* as

$$r_j = \frac{1}{|\mathcal{D}^{\mathrm{tool}}|} \sum_{x \in \mathcal{D}^{\mathrm{tool}}} \mathbf{z}(x)[j] - \frac{1}{|\mathcal{D}^{\mathrm{nontool}}|} \sum_{x \in \mathcal{D}^{\mathrm{nontool}}} \mathbf{z}(x)[j]. \tag{5}$$

Second, we measure how strongly a neuron responds to specific tool-calling behaviors. If $a_{c,j}$ is neuron j's mean activation in category c, its *enrichment* is

$$e_j = \max_c \frac{a_{c,j} - a_j}{\sigma_j}. \tag{6}$$

These categories group examples by tool-calling behavior, including parallel calls, calls with many arguments, rejecting irrelevant tools, and using tool output to generate a response. The assignment rules are in Section A.2.

Third, we penalize associations with surface structure and data sources. We compute the strongest absolute correlation between its activation and format metadata m_q, $c_j = \max_q |\rho(\mathbf{z}(\cdot)[j], m_q)|$, and the largest standardized difference between its mean activation on one data source and its overall mean, $b_j = \max_s (a_{s,j} - a_j)/\sigma_j$. Their maximum $u_j = \max(c_j, b_j)$ forms a *shortcut penalty*. After standardizing the three statistics across neurons, the validation score is

$$s_j = \widehat{r}_j + \alpha\big(\widehat{e}_j - \widehat{u}_j\big). \tag{7}$$

Relevance measures association with tool calling, enrichment measures specificity to tool-calling categories, and the shortcut penalty discounts associations with format and source metadata.

Token-level diagnosis. The second step in Figure 2-b locates the responses of each high-scoring candidate neuron by inspecting its top-activated tokens. Tokens are assigned to format regions $\mathcal{R}_{\mathrm{fmt}}$ (structural punctuation, standard schema text, special tokens, and call delimiters) or content regions $\mathcal{R}_{\mathrm{cont}}$ (user intent, tool semantics, and argument values). We remove neuron j when

$$q_j = \frac{|\{t \in \mathrm{TopTokens}(j) : t \in \mathcal{R}_{\mathrm{fmt}}\}|}{|\,\mathrm{TopTokens}(j)|} > \gamma. \tag{8}$$

Finally, the k_f highest-scoring retained neurons form $\mathcal{N}_{\mathrm{tool}}$ for sample scoring.

2.4 CAPABILITY-GUIDED DATA SELECTION

As shown in Figure 2-c, we retain each sample's activations over $\mathcal{N}_{\text{tool}}$ and combine them with neuron weights to rank candidates. For example x_i, the score is

$$f(x_i) = \sum_{j \in \mathcal{N}_{\text{tool}}} w_j \mathbf{z}(x_i)[j], \qquad w_j = s_j \ \text{IDF}(j), \qquad \text{IDF}(j) = \log \frac{1 + |\mathcal{D}^{\text{pool}}|}{1 + \text{df}(j)} + 1, \tag{9}$$

where $\text{df}(j)$ counts pool examples that activate neuron j. The validation score measures capability relevance; IDF reduces the weights of neurons activated by many examples, since they are less useful for distinguishing candidates. TOOLSAE returns the k highest-scoring examples,

$$\mathcal{D}^{\text{sel}} = \text{TopK}_{x \in \mathcal{D}^{\text{pool}}} f(x), \tag{10}$$

and mixes them with the same $\mathcal{D}^{\text{gen}}$ used by every selection baseline. The final model is trained from the original base checkpoint rather than the warm-up checkpoint used for neuron identification, so the comparison measures the effect of data selection rather than warm-up training. Algorithmic details, thresholds, and the number of neurons retained at each filtering step are provided in Section A.5.

3 EXPERIMENTS

To evaluate TOOLSAE, we conduct a comprehensive experimental study addressing three research questions: **(RQ1)** Can TOOLSAE effectively select tool-calling training data (§ 3.2)? **(RQ2)** How do different components affect its performance (§ 3.3)? **(RQ3)** Are the identified neurons stable and interpretable (§ 3.4)?

3.1 EXPERIMENTAL SETUP

Training data. We form a pool of 135,910 tool-calling examples from xLAM (Zhang et al., 2025), Glaive[1], ToolACE (Liu et al., 2025), and When2Call (Ross et al., 2025), covering single and parallel calls, multi-turn interactions, and rejection of irrelevant tools. We format all records using the standard Qwen3.5 ChatML-style template and mix a fixed set of 32k general instructions from Tulu 3 (Lambert et al., 2024) into every SFT run. Dataset statistics are summarized in Section A.1.

Evaluation. We evaluate tool calling on BFCL v4 (Patil et al., 2025) across three main categories: (1) *Non-Live AST* uses expert-curated synthetic APIs and AST matching to assess Simple, Multiple, Parallel, and Parallel Multiple calls; (2) *Live AST* tests real APIs, for which we report the Simple and Multiple subsets; and (3) *Multi-Turn* evaluates extended interactions in Base, Missing Function, Missing Parameter, and Long Context scenarios. We also evaluate general-domain performance on MMLU (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), IFEval (Zhou et al., 2023), and HumanEval (Chen et al., 2021), covering knowledge, math, instruction following, and coding. Further evaluation details are provided in Section B.2.

Setup. We use Qwen3.5-9B-Base (Team, 2026) as the starting checkpoint for all fine-tuning runs. Neuron activations are extracted from the penultimate (second-to-last) MLP layer using a Top-K SAE (Gao et al., 2025) with expansion factor 32 ($d' = 131{,}072$) and sparsity $K = 192$. All data selection methods retain 5% of the tool pool (approximately 6.8k examples) and use the same final SFT configuration. We provide full training settings in Section A.5.

Baselines. We compare against (i) *distribution-alignment methods*: BM25 (Robertson & Zaragoza, 2009) and DSIR (Xie et al., 2023); (ii) *influence-based methods*: LESS (Xia et al., 2024) and MATES (Yu et al., 2024); and (iii) *methods based on difficulty or diversity*: IFD (Li et al., 2024a) and DIVERSITY-SAE (Yang et al., 2025). We also report RANDOM selection (averaged over three seeds), the original BASE MODEL, general-only SFT (GENERAL ONLY), and full-pool training (FULL DATA). Implementation details are provided in Section B.1.

[1] https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2

Table 1: Function-calling performance on BFCL v4. Selection methods use 5% of the tool pool and the same 32k general instructions. The highest score in each metric column across all methods is in **bold**; the second-highest distinct score is underlined, including ties.

		Non-Live AST					Live AST			Multi-Turn				
Method	**Tool data**	Simple	Multi	Para	PM	**Avg.**	Simple	Multi	**Avg.**	Base	MissF	MissP	LongCtx	**Avg.**
References														
Base Model	0	38.50	58.00	28.00	20.00	36.13	33.33	19.28	22.04	1.00	0.00	0.00	1.12	0.53
General Only	0	**74.00**	91.00	71.00	47.50	70.88	70.54	63.63	64.99	13.00	8.50	7.00	12.50	10.25
Full Data	136k	71.75	93.00	90.50	86.50	85.44	80.62	**75.50**	**76.51**	21.00	15.50	**16.00**	18.50	17.75
Data selection baselines (5%)														
Random (avg. ×3)	6.8k	71.47	93.67	90.83	86.83	85.70	82.17	70.50	72.79	19.17	10.50	11.17	16.33	14.29
BM25	6.8k	70.25	85.50	82.00	72.50	77.56	76.74	70.94	72.08	19.50	14.50	12.50	18.50	16.25
DSIR	6.8k	69.08	87.50	80.00	72.00	77.15	72.48	70.18	70.63	20.00	14.50	14.50	19.50	17.12
LESS	6.8k	68.50	83.50	83.00	72.50	76.88	75.97	66.48	68.34	17.50	12.50	11.50	15.50	14.25
MATES	6.8k	69.75	**94.50**	50.50	65.00	69.94	79.84	72.36	73.84	19.50	11.50	11.00	15.50	14.37
IFD	6.8k	66.08	90.50	59.00	78.00	73.40	71.32	65.05	66.28	18.00	11.50	12.00	15.00	14.12
Diversity-SAE	6.8k	69.00	90.00	84.00	76.50	79.88	79.84	71.42	73.08	18.50	15.00	13.00	16.00	15.62
TOOLSAE (ours)	6.8k	72.42	**94.50**	**91.00**	**90.00**	**86.98**	**84.11**	72.46	74.75	**24.50**	**17.50**	15.50	**22.00**	**19.88**

Note. Following the official BFCL v4 protocol, Non-Live AST and Multi-Turn Avg. are equal-weight averages. Live AST Avg. is weighted by the number of examples in each category.

3.2 MAIN RESULTS

To evaluate the effectiveness of TOOLSAE **(RQ1)**, we report results across structured calls, real-API calls, and multi-turn interactions on BFCL v4 (Table 1), alongside general-domain benchmarks (Table 2). The results lead to the following observations.

Comparison with data selection baselines. On Non-Live AST, TOOLSAE achieves the best average among data selection methods, slightly above Random, while IFD and Diversity-SAE both perform worse than Random (Table 1). Repeated JSON structures and function schemas can make useful calls appear easy to a difficulty criterion such as IFD; global SAE-neuron coverage can likewise reward formatting responses rather than tool-calling behaviors. Target-guided methods also vary across evaluation groups, consistent with the difficulty of covering tool-calling behaviors using a small reference set. On Multi-Turn, TOOLSAE gains 5.59 points over Random and leads all selection baselines across ordinary and more complex interactions, suggesting that the identified neurons help prioritize traces with multi-turn capability signals. Its smaller advantage on Live AST may reflect the wider variety of real APIs.

Selected data versus the full pool. Full Data uses all 136k tool-calling examples, yet it does not consistently outperform a 5% random subset: Random scores higher on Non-Live AST Multiple, Parallel, and Parallel Multiple, as well as the group average (Table 1). This pattern suggests that adding more traces can introduce redundant or conflicting training signals rather than uniformly improving structured tool calling. By prioritizing examples that strongly activate tool-capability neurons, TOOLSAE further improves on both Random and Full Data on Non-Live AST and Multi-Turn. Full Data remains stronger on Live AST, where broader coverage of real APIs may still help.

General-domain performance. Table 2 shows that TOOLSAE maintains general-domain performance while improving tool calling. Its scores on MMLU, GSM8K, IFEval, and HumanEval are comparable to those of other data selection methods, and its overall average is higher than Full Data. Under the fixed general-instruction mixture, this suggests that the selected tool traces do not substantially interfere with the model's broader capabilities.

3.3 ABLATION STUDIES

To better understand which components drive the effectiveness of TOOLSAE **(RQ2)**, we ablate its sparse representation, tool-capability neuron identification, profiling criteria, token-level diagnosis, and neuron-weighting strategy.

Table 2: General-domain performance after training. The highest score in each metric column across all methods is in **bold**, including ties.

Method	MMLU	GSM8K	IFEval	HumanEval	Avg.
References					
Base Model	77.27	**95.98**	31.42	65.40	67.52
General Only	**77.37**	94.54	70.24	**71.16**	**78.33**
Full Data	76.47	93.71	62.48	69.34	75.50
Data selection baselines (5%)					
Random	77.03	91.76	71.23	69.28	77.33
BM25	77.10	85.22	71.16	68.81	75.57
DSIR	77.08	86.58	70.98	67.45	75.52
LESS	77.00	85.67	70.79	68.82	75.57
MATES	77.08	85.60	**73.01**	67.82	75.88
IFD	76.99	92.49	69.87	71.07	77.61
Diversity-SAE	77.12	90.52	70.27	69.00	76.73
TOOLSAE	76.93	91.66	71.16	70.37	77.53

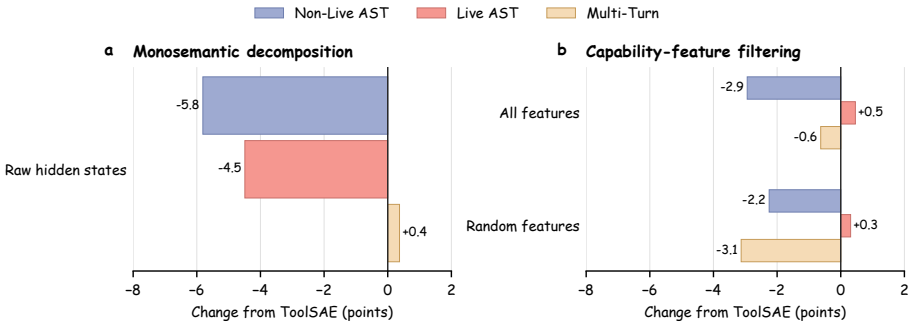

Figure 3: SAE representations and tool-capability neuron filtering. Bars report the change from full TOOLSAE (negative is worse). **(a)** Replacing SAE neuron activations with raw hidden states. **(b)** Replacing $\mathcal{N}_{\text{tool}}$ with all SAE neurons or a random neuron set of equal size.

Effect of SAE Representations and Neuron Identification. Figure 3 evaluates the roles of SAE representations and neuron identification. Raw hidden states substantially reduce Non-Live and Live AST performance, indicating that neuron identification is less effective with polysemantic representations. Scoring with all SAE neurons or a random subset lowers Non-Live AST and Multi-Turn performance, while yielding slightly higher Live AST scores. These results support the usefulness of capability-specific neuron selection, particularly for Non-Live AST and Multi-Turn tasks.

Effect of Neuron Profiling Criteria. Figure 4-a shows that the shortcut penalty is the most consistently important score component. Removing it lowers all three BFCL category averages. Enrichment primarily matters for Multi-Turn: relevance alone can identify neurons that respond more strongly to tool-calling traces, but misses the rarer interaction patterns needed for missing-function, missing-parameter, and long-context scenarios. Increasing the number of retained neurons does not always improve performance (Figure 4-b). Non-Live AST is highest at $k_f = 512$; a smaller set misses some tool-calling behaviors, whereas a larger set reintroduces irrelevant neurons. We therefore use 512 throughout.

Ablation of Token-Level Diagnosis. Skipping token-level diagnosis sharply reduces Non-Live AST performance (Figure 5-a). Expanding the unfiltered neuron set narrows but does not remove the gap. Together with the degradation caused by removing the shortcut penalty, this supports combining contrastive neuron profiling with token-level diagnosis: profiling penalizes associations with formatting and data sources, while diagnosis identifies neurons that respond mainly to syntax and delimiters.

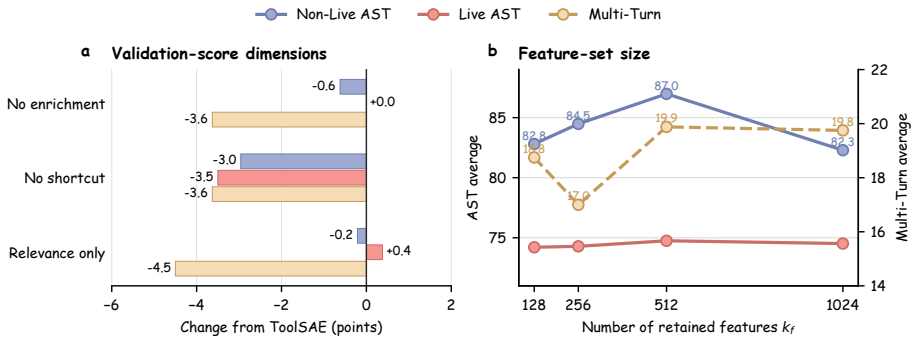

Figure 4: Ablation of contrastive neuron profiling and the number of retained capability-specific neurons. **(a)** Change from full TOOLSAE after removing enrichment or shortcut penalty, or retaining relevance alone. **(b)** Absolute BFCL category averages as k_f varies.

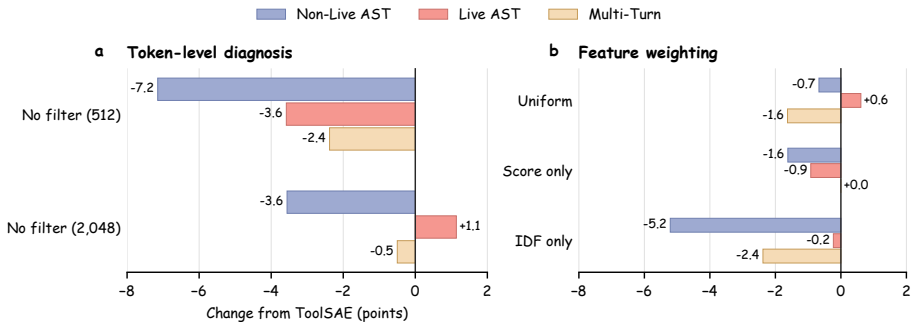

Figure 5: Token-level diagnosis and neuron weighting. Bars report the change from full TOOLSAE. **(a)** Skipping token-level format-shortcut filtering and directly using sample-level candidates. **(b)** Replacing validation-score×IDF weights with uniform, validation-only, or IDF-only weights.

Comparison of Neuron Weighting Strategies. Uniform weighting also performs well, suggesting that the retained neurons already help distinguish useful training examples. Nevertheless, the full validation-score×IDF weight is the most balanced across categories (Figure 5-b). IDF alone performs poorly because low activation frequency does not establish relevance to tool calling; the validation score provides this information. Complete category-level results are reported in Section C.2.

3.4 STABILITY AND INTERPRETABILITY ANALYSIS

Beyond downstream performance, we examine whether the identified neurons remain stable across data subsamples and inspect their token-level activations for tool-calling semantics beyond surface formatting (**RQ3**).

Stability of relevance-based feature ranking. The identified neurons show high pairwise overlap across random data subsamples (Figure 6, left). Even a small subsample recovers much of the full-data neuron set, and overlap increases with more data (right). These results suggest that neuron identification is stable across data samples.

The retained neurons capture tool-calling semantics. Representative neurons activate on meaningful parts of parameter names, physical quantities such as *mass* and *volume* across different schemas, and topic-specific words in responses based on tool output. Their strongest activations are not concentrated on braces or JSON keys. Section C.4 provides token-level examples. These qualitative examples, together with the shortcut ablations and stability results, support the finding that the retained neurons capture tool-calling semantics beyond surface formatting.

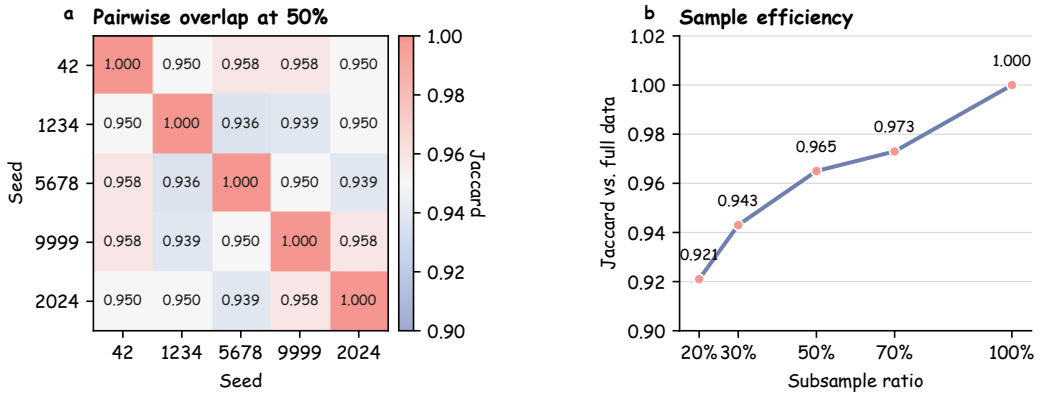

Figure 6: Stability of tool-capability neuron identification. **Left**: pairwise Jaccard overlap across five 50% subsamples. **Right**: overlap with the neuron set obtained from the full pool.

4 Related Work

Tool-calling data and selection. Tool-calling agents need training traces spanning diverse APIs and interactions. Toolformer uses self-supervised API annotations (Schick et al., 2023), while ToolLLM provides multi-step API supervision (Qin et al., 2024). APIGen and ToolACE improve synthetic data quality through execution, rule-based, and model-based verification (Liu et al., 2024b; 2025); When2Call adds examples of when to ask for clarification or avoid calling a tool (Ross et al., 2025). Yet large pools can contain repeated behaviors across different schemas. Data selection improves data efficiency by prioritizing useful examples. Existing methods prioritize difficulty, quality, or diversity (Li et al., 2024a; Liu et al., 2024a), or align examples with a target set through lexical relevance, distribution matching, or estimated training influence (Robertson & Zaragoza, 2009; Xie et al., 2023; Xia et al., 2024; Yu et al., 2024). However, generic criteria can favor surface differences, and small target sets may not cover the range of tool-calling behaviors. TOOLSAE selects examples based on how strongly they activate tool-capability neurons.

Sparse Autoencoders and Neuron Activations. SAEs reconstruct model activations from a small number of learned feature directions, helping separate concepts within polysemantic representations (Huben et al., 2024). Each latent neuron records the activation of one such feature. Top-K SAEs directly control the number of active neurons and improve the balance between reconstruction quality and sparsity (Gao et al., 2025). These activations also support data selection: Diversity-SAE promotes coverage of SAE neurons (Yang et al., 2025), MoNA aligns candidate activations with target examples (Ma et al., 2026), and FineScope retrieves domain-specific examples for pruning and fine-tuning (Bhattacharyya et al., 2025). While these methods emphasize activation coverage or target alignment, TOOLSAE explicitly identifies tool-capability neurons through contrastive neuron profiling and token-level diagnosis, then selects examples based on their weighted activations.

5 Conclusion

We present TOOLSAE, a capability-guided framework that selects tool-calling training data through tool-capability neuron activations. Sparse autoencoders provide interpretable neuron representations, while contrastive neuron profiling and token-level diagnosis identify relevant neurons and exclude formatting shortcuts. TOOLSAE ranks candidate traces by their weighted activations over these neurons. Our evaluations show that training on just 5% of the tool-calling data outperforms full-data training on key tool-calling tasks while maintaining comparable general-domain performance. Ablation and interpretability analyses support the selection design and show that the identified neurons respond to tool-calling semantics beyond surface formatting. These findings point to model-internal activations as useful signals for data-efficient tool-calling training.

AI USE STATEMENT

We used generative AI tools for language polishing, assistance with code implementation, and analysis of experimental results. The authors designed the research, conducted the experiments, validated the scientific claims and results, and reviewed all AI-assisted content. We take responsibility for the final content of this work.

ETHICS STATEMENT

This work uses publicly available datasets and does not involve human participants or the collection of private or sensitive personal data. The experiments evaluate tool-calling data selection on established benchmarks rather than deploying agents to real users. We have adhered to the ICLR Code of Ethics throughout this work.

REPRODUCIBILITY STATEMENT

An anonymous repository for this work is available at `https://anonymous.4open.science/r/ToolSAE-anonymous-ECFF`. Implementation and evaluation details are provided in the appendix.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martínez, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm2: When smol goes big—data-centric training of a fully open small language model. In *Conference on Language Modeling*, 2025. URL `https://openreview.net/forum?id=3JiCl2A14H`.

Chaitali Bhattacharyya, Hyunsei Lee, Junyoung Lee, Shinhyoung Jang, Il hong Suh, and Yeseong Kim. FineScope: SAE-guided data selection enables domain specific LLM pruning and finetuning. *arXiv preprint arXiv:2505.00624*, 2025. URL `https://arxiv.org/abs/2505.00624`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Rei Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *International Conference on Learning Representations*, volume 2025, pp. 26721–26754, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=d7KBjmI3GmQ`.

Robert Huben, Hoagy Cunningham, Logan Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *International Conference on Learning Representations*, volume 2024, pp. 7827–7845, 2024.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 14255–14273, 2024a. doi: 10.18653/v1/2024.acl-long.769.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7602–7635, 2024b. doi: 10.18653/v1/2024.naacl-long.421. URL https://aclanthology.org/2024.naacl-long.421/.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=BTKAeLqLMw.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2406.18518.

Da Ma, Gonghu Shang, Zhi Chen, Libo Qin, Yijie Luo, Hongshen Xu, Lei Pan, Shuai Fan, Kai Yu, and Lu Chen. Task-specific data selection for instruction tuning via monosemantic neuronal activations. *Advances in Neural Information Processing Systems*, 38:138006–138034, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 48371–48392, 2025.

Andrew J. Peters, Simon X. Chen, and Takaki Komiyama. Emergence of reproducible spatiotemporal activity during motor learning. *Nature*, 510:263–267, 2014. doi: 10.1038/nature13235. URL https://www.nature.com/articles/nature13235.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. When2call: When (not) to call tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3391–3409, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2302.04761.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54104–54132. PMLR, 2024. URL https://proceedings.mlr.press/v235/xia24c.html.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. Data selection for language models via importance resampling. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2302.03169.

Xianjun Yang, Shaoliang Nie, Lijuan Liu, Suchin Gururangan, Ujjwal Karn, Rui Hou, Madian Khabsa, and Yuning Mao. Diversity-driven data selection for language model tuning through sparse autoencoder. *arXiv preprint arXiv:2502.14050*, 2025. URL https://arxiv.org/abs/2502.14050.

Zichun Yu, Spandan Das, and Chenyan Xiong. MATES: Model-aware data selection for efficient pretraining with data influence models. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6gzPSMUAz2.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, et al. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597, 2025.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 400–410, 2024. doi: 10.18653/v1/2024.acl-demos.38.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

LIMITATIONS AND IMPACTS

While TOOLSAE improves the data efficiency of tool-calling training, most controlled evaluations use Qwen3.5-9B-Base and text-only instruction data. A transfer test on Qwen3-14B-Base is encouraging, but generalization across other model families remains unexplored. Extending the framework to multimodal tool-calling settings, where agents must interpret images or other inputs alongside text, is another direction for future work.

TOOLSAE offers a neuron-based perspective on data selection that may support more data-efficient tool-calling training and help analyze how examples engage model capabilities. Because tool-calling agents can act through external systems, practical deployment still requires appropriate access controls and evaluation.

A DATA AND IMPLEMENTATION DETAILS

A.1 CANDIDATE POOL

Table 3 summarizes the four sources after normalization. The sources cover different behaviors: xLAM supplies most parallel calls, Glaive supplies most conversations that use tool responses, ToolACE covers several categories at a smaller scale, and When2Call supplies rejection and non-tool examples.

Table 3: Statistics of the tool-calling candidate pool. "Reject" denotes examples that define tools but make no call.

Source	Examples	Has call	Reject	Avg. tools	Multi-turn	Parallel
xLAM	60,000	60,000	0	2.8	0	31,539
Glaive	63,358	63,358	0	1.2	62,985	0
ToolACE	10,552	9,262	1,290	3.2	800	4,125
When2Call	2,000	0	1,430	1.8	0	0
Total	135,910	132,620	2,720	2.1	63,785	35,664

A.2 TOOL-CALLING CATEGORIES

Each example is assigned to the first matching row of Table 4; the ordering prevents an example with several properties from being counted repeatedly. These categories are used only to compute neuron enrichment, not as supervision for the final model.

Table 4: Priority-ordered category assignment and resulting pool distribution.

Category	First matching condition	Count	Share
`nontool`	no call and no tools	570	0.4%
`reject_irrelevance`	no call, but tools are defined	2,720	2.0%
`tool_response_grounding`	multi-turn with a tool observation	63,753	46.9%
`multi_turn`	multi-turn without a tool observation	31	$< 0.1\%$
`parallel_call`	more than one call in one message	35,604	26.2%
`multi_tool_selection`	≥ 4 tools and ≤ 1 call	11,751	8.6%
`arg_heavy`	≥ 4 total or ≥ 3 required parameters	12,604	9.3%
`schema_rich`	serialized schema length ≥ 3000	0	0.0%
`simple_call`	none of the above	8,877	6.5%

A.3 SHORTCUT METADATA AND TOKEN REGIONS

The sample-level shortcut penalty uses six surface statistics: serialized schema length, number of tools, total and required parameter counts, total message-content length, and first-user-query length. For token-level diagnosis, a token belongs to $\mathcal{R}_{\text{fmt}}$ if it is (i) a call delimiter or function/argument marker, (ii) inside a tool schema in the system prompt, (iii) JSON structural punctuation, (iv) a

format keyword such as `arguments`, `properties`, `type`, or `required`, or (v) a special token matching the model's delimiter pattern. Other tokens, including tool names, argument values, and user-intent spans, belong to $\mathcal{R}_{\text{cont}}$.

Table 5: Candidate-pool statistics for the metadata used in the format-shortcut penalty. Lengths are measured in characters.

Attribute	Mean	Std. dev.	Median	Maximum
Schema length	988.5	856.3	660	9,820
Number of tools	2.1	1.5	2	9
Total parameters	4.4	4.2	3	66
Required parameters	3.2	2.7	2	41
Total message length	454.6	368.8	351	13,537
First user query length	107.3	104.6	84	6,727

Table 6 specifies how each statistic is computed from the tool definitions and messages.

Table 6: Definitions of the six format-shortcut metadata attributes.

Attribute	Source	Computation
Schema length	Tools	Character length of serialized tool definitions
Number of tools	Tools	Number of tools defined in the system prompt
Total parameters	Tools	Number of parameters across all defined tools
Required parameters	Tools	Number of required parameters across all defined tools
Total message length	Messages	Sum of the character lengths of all message contents
First user query length	Messages	Character length of the first user message

Table 7 gives the token-level rules used to distinguish formatting from content. A token is assigned to the format region if it matches any listed rule; otherwise it is assigned to $\mathcal{R}_{\text{cont}}$.

Table 7: Token-level rules for the format region $\mathcal{R}_{\text{fmt}}$.

Format category	Rule or example
Call markers	Tool-call delimiters and function or argument markers
Tool schema	Tokens inside tool definitions in the system prompt
JSON punctuation	Structural characters such as braces, brackets, colons, commas, and quotes
Format keywords	Tokens containing `arguments`, `properties`, `type`, or `required`
Special delimiters	Tokens matching the model's <| . . . |> or <. . . > delimiter patterns

A.4 Source and Tool-Calling Category Distribution

Table 8 shows how the candidate sources contribute to each populated category. The `multi_turn` and `schema_rich` categories, with 31 and zero examples respectively, are omitted from the cross-tab for space; they remain included in the pool totals in Table 4. The counts clarify why source-specific formatting can be a shortcut: some tool-calling behaviors are concentrated in one source.

A.5 Implementation and Hyperparameters

The warm-up and final runs use the standard Qwen3.5 ChatML-style template with thinking disabled. Warm-up uses a random 5% tool subset plus the fixed 32k general instructions to give the model an initial tool-calling capability. Final SFT always restarts from Qwen3.5-9B-Base and combines the selected tool data with the same general instructions. We use full-parameter training for one epoch, learning rate 10^{-5}, cosine decay, global batch size 128, maximum sequence length 8192, bf16, and DeepSpeed ZeRO-3, implemented with LLaMA-Factory (Zheng et al., 2024).

Table 8: Candidate examples by source and tool-calling category. "Response" denotes tool-response grounding and "reject" denotes rejection of an irrelevant tool.

Category	xLAM	Glaive	ToolACE	When2Call	Total
Simple	7,221	282	1,374	0	8,877
Parallel	31,539	0	4,065	0	35,604
Multi-tool	9,961	0	1,790	0	11,751
Arg-heavy	11,279	91	1,234	0	12,604
Response	0	62,956	797	0	63,753
Reject	0	0	1,290	1,430	2,720
Non-tool	0	0	0	570	570
Shown total	60,000	63,329	10,550	2,000	135,879

We train the Top-K SAE on the warm-up model's penultimate MLP layer using `sparsify`.[2] SAE training uses roughly 10M general-text examples from SmolLM2 (Allal et al., 2025), with expansion factor 32 ($d' = 131{,}072$) and $K = 192$.

For tool-capability neuron identification, we set $\alpha = 0.5$ in Equation (7). The filters in contrastive neuron profiling retain neurons with activation rate in $[0.001, 0.95]$, enrichment at least 0.1, and shortcut penalty at most 2.0; neurons most specific to the `nontool` or `multi_turn` categories are removed because those categories are either negative or too small for reliable estimation. We retain the top $k_n = 2{,}048$ candidates by validation score.

Token-level diagnosis uses 768 examples sampled proportionally across categories. A candidate is removed if it has more than 20 diagnostic hits and over $\gamma = 0.8$ of its top-activated tokens fall in $\mathcal{R}_{\text{fmt}}$. This removes 592 of the 2,048 candidates. We retain the top $k_f = 512$ of the 1,456 remaining neurons as $\mathcal{N}_{\text{tool}}$ and select the highest-scoring 5% of the candidate pool.

Table 9: Number of neurons retained at each filtering step.

Stage	Operation	Removed	Remaining
Initial	all SAE neurons	–	131,072
Sample-level	hard filters + top-2,048 ranking	129,024	2,048
Token-level	$> 80\%$ format-region activations	592	1,456
Final	top-512 by validation score	944	512

B Baselines and Evaluation

B.1 Baseline Implementations

Shared reference set. The four target-guided methods, BM25, DSIR, LESS, and MATES, use the same held-out 120-example reference set, constructed entirely from the 135,910 records in `candidate_pool.jsonl`. We fix the sampling seed to 42 and allocate 15 examples to each of the eight structural categories in Table 10. These labels derived from the candidate pool cover call structure, interaction context, and cases with no tool call. The resulting reference set contains 59 ToolACE, 37 xLAM, 19 Glaive, and 5 When2Call examples.

Sampling and candidate exclusion. We fill the scarcest categories first and cycle through available data sources within each category. This balances structural coverage while maintaining source diversity. An example satisfying several criteria occupies only one category's quota. Within the reference set, normalized user questions and tool-schema sets are each required to be unique. The reference examples and associated examples are removed from the selection candidates before running the four target-guided baselines.

[2] `https://github.com/EleutherAI/sparsify`

Table 10: Structural categories for reference-set sampling. Each category contributes 15 examples.

Category	Structural criterion
Single-tool call	One tool available and one call
Tool selection	One tool called from multiple available tools
Repeated-function calls	Multiple calls to the same function
Multi-function calls	Calls to different functions
Multi-turn dialogue	Multiple user turns
Tool-result context	Conversation containing tool-return context
Long-context proxy	Combined context and schema length $\geq 4{,}000$ characters
No tool call	No tool call

Distribution alignment. BM25 (Robertson & Zaragoza, 2009) uses lexical relevance to retrieve candidate examples against target examples. DSIR (Xie et al., 2023) represents source and target data with n-gram features and uses importance resampling to align their distributions. These methods select data through properties of the source and target text.

Influence-based selection. LESS (Xia et al., 2024) uses gradient alignment to estimate how training on a candidate example affects the target objective. MATES (Yu et al., 2024) learns a data-influence model to predict sample utility for the target objective. Both use model-dependent estimates of training impact, complementing the text-based alignment methods above.

IFD. Following Superfiltering (Li et al., 2024a), we score each assistant turn by the ratio of conditional to unconditional average token loss. The conditional context contains the system prompt, tool definitions, and conversation history rendered exactly as in SFT; the unconditional loss is computed from the response prefix alone. Multi-turn scores are averaged over assistant turns, and the highest-scoring examples are selected. Specifically, for assistant turn y_t with $|y_t|$ tokens, the score is

$$L_{\text{cond}}(y_t) = -\frac{1}{|y_t|}\sum_i \log p(y_{t,i} \mid \text{context}, y_{t,<i}), \tag{11}$$

$$L_{\text{alone}}(y_t) = -\frac{1}{|y_t|}\sum_i \log p(y_{t,i} \mid y_{t,<i}), \qquad \text{IFD}_t = \frac{L_{\text{cond}}(y_t)}{L_{\text{alone}}(y_t)}. \tag{12}$$

Both losses are evaluated with the unmodified Qwen3.5-9B-Base model; for R assistant turns, $\text{IFD}(x) = R^{-1}\sum_{t=1}^{R} \text{IFD}_t$.

Diversity-SAE. We compute SAE neuron activations with the same warm-up model and SAE used by TOOLSAE. For example x, $F(x)$ contains the neurons whose activation exceeds 1.0 on at least one token. Selection greedily maximizes

$$\Delta(x) = \sum_{j \in F(x)} \frac{\text{IDF}(j)}{(1+c_j)^{\beta}}, \qquad \beta = 1, \tag{13}$$

where c_j is the number of already selected examples that activate neuron j. The weight decreases as c_j grows, reducing the reward for neurons already covered by selected examples. Unlike TOOLSAE, this objective rewards global SAE diversity without distinguishing tool-capability neurons from formatting shortcuts. Here $\text{IDF}(j) = \log\big((1+N)/(1+\text{df}(j))\big) + 1$, where N is the candidate-pool size and $\text{df}(j)$ counts examples that activate neuron j. After choosing an example, the algorithm increments c_j for all $j \in F(x)$ and repeats until reaching the data budget. This IDF-weighted, diminishing-return implementation avoids the rapid saturation of strict binary coverage.

B.2 EVALUATION PROTOCOL

For BFCL v4 (Patil et al., 2025), Non-Live AST uses expert-curated synthetic APIs and includes Simple, Multiple, Parallel, and Parallel Multiple calls. Live AST uses real APIs; we evaluate Simple and Multiple, excluding its Parallel and Parallel Multiple subsets because their small sizes (16 and

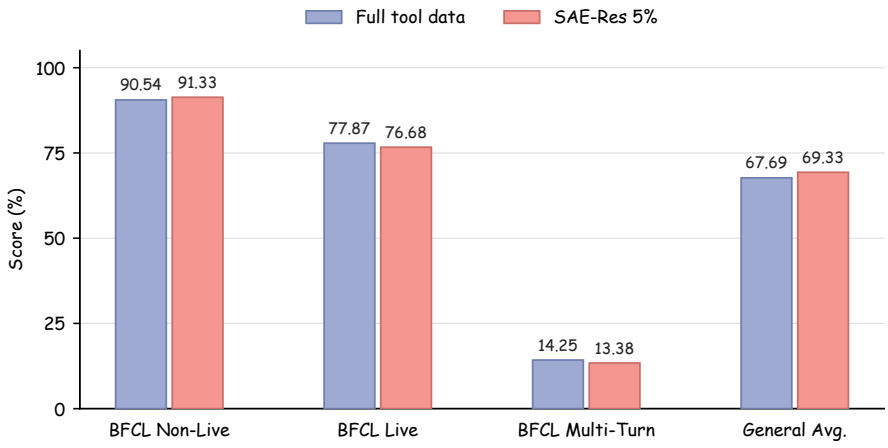

Figure 7: Cross-model transfer to Qwen3-14B-Base. The 5% subset was selected using Qwen3.5-9B and is labeled SAE-Res in the experiment records. General Avg. averages MMLU, GSM8K, IFEval, and HumanEval pass@2. The full-data run used global batch size 128 for one epoch; the 5% run used global batch size 64 for two epochs.

24 examples, respectively) lead to high score variability. Multi-Turn covers ordinary interaction, missing functions, missing parameters, and long contexts. We evaluate general knowledge, mathematical reasoning, instruction following, and code generation on MMLU, GSM8K, IFEval, and HumanEval, respectively.

Metric aggregation. We follow the official BFCL v4 aggregation protocol (Patil et al., 2025). Non-Live Simple is the equal-weight average of the Python, Java, and JavaScript scores. Non-Live Avg. equally averages Simple, Multiple, Parallel, and Parallel Multiple; Multi-Turn Avg. equally averages Base, Missing Function, Missing Parameter, and Long Context. Live Avg. weights the evaluated categories by their sample counts.

C Additional Results and Analyses

C.1 Cross-Model Transfer

We test whether the data selected using Qwen3.5-9B transfers to a different training backbone. We take the 5% tool-calling subset selected by ToolSAE with the Qwen3.5-9B warm-up model and SAE, without repeating neuron identification or data selection on the target backbone, and fine-tune Qwen3-14B-Base on this subset together with the same 32k general instructions. We compare it with Qwen3-14B-Base fine-tuned on the full tool pool and the same general instructions. This evaluates reuse of the selected examples across models, rather than transfer of SAE neurons themselves.

As shown in Figure 7, the transferred 5% subset remains close to full-data training across the three BFCL groups: it scores 91.33 versus 90.54 on Non-Live AST, 76.68 versus 77.87 on Live AST, and 13.38 versus 14.25 on Multi-Turn. Its general-domain average is 69.33 versus 67.69 for full data. Thus, the selected examples retain useful tool-calling signal for Qwen3-14B-Base while using only about one twentieth as many tool examples. The gains are not uniform, and the two runs differ in batch size and epoch count; this result is evidence of promising cross-model reuse, not a controlled estimate of the effect of data selection alone.

C.2 Full Ablation Results

Table 11 reports the category averages used to produce Figures 3 to 5. Live AST averages weight its Simple and Multiple subsets by their 258 and 1,053 examples, respectively. All models use the same 5% data budget and training setup.

Table 11: Ablation results on the three BFCL v4 category averages. The highest score in each metric column is in **bold**, including ties.

Component	Setting	Non-Live AST	Live AST	Multi-Turn
Full method	TOOLSAE	**86.98**	74.75	19.88
Representation	`Raw hidden states`	81.17	70.26	**20.25**
Neuron set	`Random neurons`	84.73	75.06	16.75
	`All neurons`	84.04	75.21	19.25
Validation score	`Relevance only`	86.77	75.13	15.38
	`No shortcut penalty`	84.02	71.25	16.25
	`No enrichment`	86.36	74.75	16.25
	`Equal score weights` ($\alpha = 1$)	83.56	72.16	15.13
Token diagnosis	`No filter` ($k_f = 512$)	79.83	71.17	17.50
	`No filter` ($k_f = 2{,}048$)	83.42	**75.89**	19.38
Neuron count	$k_f = 128$	82.81	74.21	18.75
	$k_f = 256$	84.48	74.29	17.00
	$k_f = 1{,}024$	82.29	74.52	19.75
Weight	`Uniform`	86.31	75.36	18.25
	`Validation score only`	85.36	73.83	19.88
	`IDF only`	81.77	74.52	17.50

Non-Live AST performance first increases and then decreases as more neurons are retained: too few miss some tool-calling behaviors, while too many include irrelevant neurons. Live AST changes less across these settings. Multi-Turn performance remains similar at $k_f \geq 512$, supporting our choice of 512 neurons.

Tables 12 to 14 give the underlying BFCL v4 subcategory scores for every setting in Table 11. The Live AST averages use the same sample-count weighting as the main results; all other category averages follow the equal-weight protocol in Section B.2.

Table 12: Detailed Non-Live AST ablation results. PM denotes Parallel Multiple.

Setting	Simple	Multiple	Parallel	PM	Avg.
TOOLSAE	72.42	94.50	91.00	90.00	86.98
Raw hidden states	69.67	90.50	85.50	79.00	81.17
Random neurons	70.92	92.50	89.50	86.00	84.73
All neurons	71.17	92.00	89.00	84.00	84.04
Relevance only	72.08	95.00	91.50	88.50	86.77
No shortcut penalty	71.08	94.00	86.50	84.50	84.02
No enrichment	71.92	95.00	90.50	88.00	86.36
Equal score weights ($\alpha = 1$)	71.25	91.50	89.00	82.50	83.56
No filter ($k_f = 512$)	70.33	88.50	85.50	75.00	79.83
No filter ($k_f = 2{,}048$)	71.17	92.50	87.50	82.50	83.42
$k_f = 128$	69.25	91.00	87.50	83.50	82.81
$k_f = 256$	71.92	92.00	89.00	85.00	84.48
$k_f = 1{,}024$	69.67	92.50	88.00	79.00	82.29
Uniform neuron weights	71.75	95.00	92.50	86.00	86.31
Validation score only	71.42	93.50	91.50	85.00	85.36
IDF only	69.58	92.50	88.00	77.00	81.77

Table 13: Detailed Live AST ablation results. Avg. weights Simple and Multiple by 258 and 1,053 examples, respectively.

Setting	Simple	Multiple	Avg.
TOOLSAE	84.11	72.46	74.75
Raw hidden states	79.46	68.00	70.26
Random neurons	81.01	73.60	75.06
All neurons	84.11	73.03	75.21
Relevance only	84.11	72.93	75.13
No shortcut penalty	82.95	68.38	71.25
No enrichment	84.50	72.36	74.75
Equal score weights ($\alpha = 1$)	82.95	69.52	72.16
No filter ($k_f = 512$)	84.11	68.00	71.17
No filter ($k_f = 2{,}048$)	84.11	73.88	75.89
$k_f = 128$	84.88	71.60	74.21
$k_f = 256$	82.95	72.17	74.29
$k_f = 1{,}024$	83.33	72.36	74.52
Uniform neuron weights	85.66	72.84	75.36
Validation score only	82.95	71.60	73.83
IDF only	81.40	72.84	74.52

Table 14: Detailed Multi-Turn ablation results. MissF and MissP denote Missing Function and Missing Parameter; LongCtx denotes Long Context.

Setting	Base	MissF	MissP	LongCtx	Avg.
TOOLSAE	24.50	17.50	15.50	22.00	19.88
Raw hidden states	24.50	20.00	14.00	22.50	20.25
Random neurons	21.50	16.50	10.50	18.50	16.75
All neurons	25.00	15.00	15.00	22.00	19.25
Relevance only	19.50	14.00	12.00	16.00	15.38
No shortcut penalty	21.50	14.50	12.50	16.50	16.25
No enrichment	21.00	12.50	14.00	17.50	16.25
Equal score weights ($\alpha = 1$)	18.50	13.50	14.00	14.50	15.13
No filter ($k_f = 512$)	20.50	15.50	14.50	19.50	17.50
No filter ($k_f = 2{,}048$)	23.50	16.00	18.00	20.00	19.38
$k_f = 128$	23.50	15.00	16.00	20.50	18.75
$k_f = 256$	22.00	15.00	15.00	16.00	17.00
$k_f = 1{,}024$	24.00	18.00	15.00	22.00	19.75
Uniform neuron weights	22.50	16.00	15.00	19.50	18.25
Validation score only	26.00	17.50	15.50	20.50	19.88
IDF only	23.00	16.00	12.50	18.50	17.50

C.3 Stability Protocol

For random-seed stability, we subsample 50% of the activation records with seeds 42, 1234, 5678, 9999, and 2024, rank neurons using relevance, and compute pairwise Jaccard overlap between the top-512 sets. All pairwise overlaps are at least 0.936, with mean approximately 0.95. To test stability with less data, we fix the seed, vary the retained fraction from 20% to 100%, and compare each top-512 set with the full-data set; the overlaps are 0.921, 0.943, 0.965, 0.973, and 1.000.

C.4 Interpretability Cases

We inspect high-activation contexts for neurons that survive both filters. Figures 8 to 10 present one representative message for each of three neurons. The displayed messages retain the tool definitions and all conversation turns; only repeated tool-call format instructions are abbreviated with ellipses. Neuron 99,314 responds to semantic suffixes in parameter names, rather than surrounding schema punctuation. Neuron 122,174 responds to physical-quantity fields such as `mass` and `volume`. Neuron 10,822 responds to `Index` in an assistant reply grounded in a BMI tool result. Orange highlights indicate the reported high-activation spans, without encoding activation magnitude.

<|im_start|>system

Tools You have access to the following functions: <tools>
{"type": "function", "function": {"name": "potential_energy", "description":
"Calculates the electrostatic potential energy given the charge and voltage.",
"parameters": {"type": "dict", "properties": {"charge": {"description": "The
electric charge.", "type": "float"}, "voltage": {"description": "The electric
potential difference (voltage).", "type": "float"}}, "required": ["charge",
"voltage"]}}}
{"type": "function", "function": {"name": "dice_roll_probability",
"description": "Calculates the probability of rolling a specific sum with a
given number of dice, each having a certain number of faces.", "parameters":
{"type": "dict", "properties": {"target_sum": {"description": "The target sum to
 calculate the probability for.", "type": "integer", "default": 6}, "num_dice":
{"description": "The number of dice being rolled.", "type": "integer",
"default": 6}, "num_faces": {"description": "The number of faces on each die.
Defaults to 6.", "type": "integer"}}, "required": ["target_sum", "num_dice"]}}}
{"type": "function", "function": {"name": "batting_average", "description":
"Calculates the batting average of a baseball player based on the number of hits
 and at-bats.", "parameters": {"type": "dict", "properties": {"num_hits":
{"description": "The number of hits.", "type": "integer", "default": 3},
"num_at_bats": {"description": "The number of at-bats.", "type": "integer",
"default": 3}, "precision": {"description": "The number of decimal places to
round the batting average. Defaults to 3.", "type": "integer"}}, "required":
["num_hits", "num_at_bats"]}}}
</tools> If you choose to call a function ONLY reply in the following format
with NO suffix: <tool_call> <function=...> <parameter=...> ... </parameter>
</function> </tool_call> <IMPORTANT> Function calls MUST follow this format;
required parameters MUST be specified; ... </IMPORTANT>
<|im_end|>

<|im_start|>user

Calculate the potential energy for a charge of 2 Coulombs at a voltage of 150
volts.
<|im_end|>

<|im_start|>assistant

<think></think> <tool_call> <function=potential_energy> <parameter=charge> 2
</parameter> <parameter=voltage> 150 </parameter> </function> </tool_call>
<|im_end|>

Figure 8: Neuron 99,314: activation on *semantic suffixes of parameter names* in the complete tool schema and conversation for sample `xlam_009712`.

<|im_start|>system

Tools You have access to the following functions: <tools>
{"type": "function", "function": {"name": "density", "description": "Calculates the density of an object given its mass and volume.", "parameters": {"type": "dict", "properties": {"mass": {"description": "The mass of the object.", "type": "float"}, "volume": {"description": "The volume of the object.", "type": "float"}}, "required": ["mass", "volume"]}}}
{"type": "function", "function": {"name": "find_longest_palindromic_substring", "description": "Finds the longest palindromic substring in a string.", "parameters": {"type": "dict", "properties": {"s": {"description": "The input string.", "type": "string"}}, "required": ["s"]}}}
</tools> If you choose to call a function ONLY reply in the following format with NO suffix: <tool_call> <function=...> <parameter=...> ... </parameter> </function> </tool_call> <IMPORTANT> Function calls MUST follow this format; required parameters MUST be specified; ... </IMPORTANT>
<|im_end|>

<|im_start|>user

Find the longest palindromic substring in the phrase 'Eva, can I see bees in a cave?'.
<|im_end|>

<|im_start|>assistant

<think></think> <tool_call> <function=find_longest_palindromic_substring> <parameter=s> Eva, can I see bees in a cave? </parameter> </function> </tool_call>
<|im_end|>

Figure 9: Neuron 122,174: activation on *physical-quantity fields* in the tool schema for sample `xlam_009547`.

<|im_start|>system

Tools You have access to the following functions: <tools>
{"type": "function", "function": {"name": "calculate_bmi", "description":
"Calculate the Body Mass Index (BMI) of a person", "parameters": {"type":
"dict", "properties": {"weight": {"description": "The weight of the person in
kilograms", "type": "float"}, "height": {"description": "The height of the
person in meters", "type": "float"}}, "required": ["weight", "height"]}}}
</tools> If you choose to call a function ONLY reply in the following format
with NO suffix: <tool_call> <function=...> <parameter=...> ... </parameter>
</function> </tool_call> <IMPORTANT> Function calls MUST follow this format;
required parameters MUST be specified; ... </IMPORTANT> You are a helpful
assistant
<|im_end|>

<|im_start|>user

Hi, I would like to know my BMI. I weigh 70 kilograms and my height is 1.75
meters.
<|im_end|>

<|im_start|>assistant

<think></think> <tool_call> <function=calculate_bmi> <parameter=weight> 70
</parameter> <parameter=height> 1.75 </parameter> </function> </tool_call>
<|im_end|>

<|im_start|>user

<tool_response> {"bmi": 22.86} </tool_response>
<|im_end|>

<|im_start|>assistant

<think></think> Based on the information provided, your Body Mass Index (BMI) is
 22.86. This is considered a healthy weight range. However, it's always a good
idea to consult with a healthcare professional for personalized advice.
<|im_end|>

Figure 10: Neuron 10,822: activation on *the domain term Index* in the assistant reply after a BMI tool response in sample `glaive_029084`.