A Comparative Analysis of Sparse Autoencoder and Activation Difference in Language Model Steering

Anonymous ACL submission

Abstract

Sparse autoencoders (SAEs) have recently emerged as a powerful tool for language model steering. Prior work has explored top-k SAE latents for steering, but we observe that many dimensions among the top-k latents capture non-semantic features such as punctuation rather than semantic attributes like instructions. To address this, we propose focusing on a single, most relevant SAE latent (top-1), eliminating redundant features. We further identify a limitation in constant SAE steering, which often produces degenerate outputs such as repetitive single words. To mitigate this, we introduce a token-wise decaying steering strategy, enabling more faithful comparisons with mean activation difference baselines. Empirically, we show that steering an SAE latent associated with reasoning reliably elicits step-by-step mathematical reasoning and enhances inference quality, functionally resembling the effect of appending a guiding token. Our results demonstrate that SAEs outperform mean activation difference methods on mathematical reasoning benchmarks and match their performance on IF-Eval.

1 Introduction

Steering language models by directly manipulating internal activations is an increasingly popular method for controlling model behavior without fine-tuning or prompt engineering (Stolfo et al., 2024; Postmus and Abreu, 2024) . Methods such as mean activation difference (**MeanActDiff**) (Stolfo et al., 2024) and sparse autoencoder (**SAE**) (Makelov, 2024; Mayne et al., 2024; O'Brien et al., 2024)-based steering have shown initial success in tasks including sentiment control (Tigges et al., 2024) and style transfer (Turner et al., 2023; Liu et al., 2024b; Scalena et al., 2024; von Rütte et al., 2024). Recently, some works have extended the scenarios to complex math reasoning and instruction-following tasks (Galichin et al., 2025; Wu et al., 2025b), demonstrating the robustness and versatility of SAE-based steering.

However, MeanActDiff lacks interpretability, offering little insight into the manipulated semantic features (Liu et al., 2024b). SAE-based steering provides a more interpretable alternative by exposing sparse, human-understandable activation features (Joseph Bloom and Chanin, 2024). Existing SAE steering methods often introduce noise and redundancy by selecting top-activated features without verifying their relevance (Shu et al., 2025; Rajamanoharan et al., 2024). In addition, most existing methods apply a constant steering strength throughout generation, which risks destabilizing generation or causing unnatural repetition, particularly in multi-step reasoning tasks, as shown in Figure 1. These limitations have prevented steering methods from being broadly adopted in more challenging scenarios, such as chain-of-thought math reasoning (Wei et al., 2022) and fine-grained instruction (Zhou et al., 2023) adherence, where precise and stable control is essential.

To address these challenges, we first propose an SAE feature selection mechanism based on subtracting top-activated feature sets, which isolates the dimensions most responsible for behavioral shifts while filtering out irrelevant activations. Second, we design a token-wise decaying steering strategy that dynamically reduces intervention strength during generation, improving stability and mitigating oversteering. We evaluate these methods on Gemma-2-2b and Gemma-2-9b for chain-of-thought math reasoning (GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MAWPS (Koncel-Kedziorski et al., 2016), ASDIV (Miao et al., 2020)) and on Gemma-2-9b-it for instruction following (IFEval (Zhou et al., 2023), covering JSON formatting, case sensitivity, multilingual responses, and lexical constraints). Our results show that decaying SAE steering im-

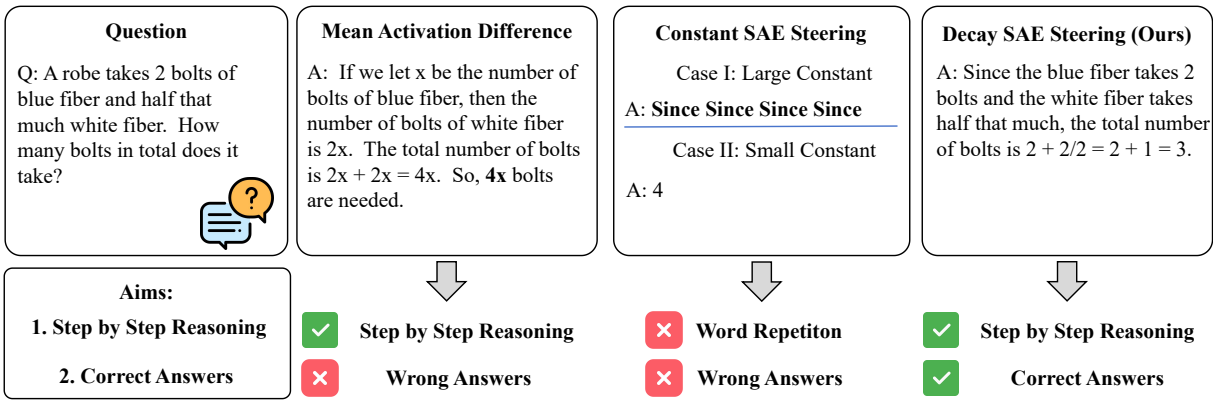


Figure 1: Mean Activation Difference (MeanActDiff) produces incorrect answers in math reasoning cases, while constant SAE steering often leads to word repetitions or wrong outputs. Our proposed decaying strategy applies the SAE steering vector progressively, yielding both step-by-step reasoning and correct answers, and enabling a fair comparison with MeanActDiff.

proves stability and achieves finer control compared to both MeanActDiff and constant SAE steering. In math reasoning, SAE steering reliably elicits step-by-step solutions without requiring few-shot prompts, outperforming MeanActDiff. In instruction following, our feature selection method more clearly identifies redundant latents, while in math reasoning further refinement may be needed. Finally, we observe that SAE steering behaves similarly to appending an additional guiding token to the prompt.

2 Related Works

Language Model Steering via Activation Editing Recent research into mechanistic interpretability explores direct control over language model outputs by intervening in their internal workings (Conmy et al., 2023; Rai et al., 2024). This is achieved by adding or subtracting steering vectors from hidden activations at specific layers to influence the style, tone, or content of the generated output (Subramani et al., 2022; Chalnev et al., 2024). A primary method for identifying these directions is the Mean Activation Difference (MeanActDiff) (Stolfo et al., 2024), which computes the average difference in activations between input pairs that elicit contrasting behaviors (e.g., generic vs. instruction-following responses) (Liu et al., 2024a). Several variants such as MeanActDiff-PCA and Linear Artificial Tomography (LAT) have been developed (Zou et al., 2025; Wu et al., 2025b).

Sparse Autoencoders for Intepretable Steering To overcome the lack of interpretability in MeanActDiff's global steering vectors, Sparse Autoencoders (SAEs) offer a more expressive alternative by learning a compressed and disentangled representation of a model's hidden activations (Lieberum et al., 2024; Gao et al., 2024). SAE training has followed diverse directions. Some works like Gated, Top-K and Switch SAE are trying to improve the SAE architecture (Gao et al., 2024; Rajamanoharan et al., 2024; Mudide et al., 2024), while others like LayerGroup, and Formal Language SAE (Ghilardi et al., 2024; Menon et al., 2024) are trying to improve the training strategies. These pre-trained SAEs (Lieberum et al., 2024; He et al., 2024; Chaudhary and Geiger, 2024) have enabled applications such as steering semantic instructions (Wu et al., 2025a), biological features (Farrell et al., 2024), reveal the hallucinations in LLMs (Ferrando et al., 2024), and study cross-lingual toxicity behaviours (Gallifant et al., 2025).

3 Methods

3.1 Preliminaries and Notation

Let $\mathcal{T} = \{\mathcal{T}_i\}_{i=0}^{K-1}$ denote a test set comprising K instances. Correspondingly, let $\mathcal{T}^* = \{\mathcal{T}_i^*\}_{i=0}^{K-1}$ be an augmented version of this set, where each instance is modified to elicit a desired behavior. In the context of mathematical reasoning, this augmentation is primarily achieved through in-context learning (ICL), where few-shot chain-of-thought examples are prepended to the input. For instruction following cases, we appended the instructions to the input as well. Let $\mathcal{F}$ represent a pretrained large language model. We define $\mathcal{A}_l(\cdot)$ as an operator that extracts the hidden activation vector (e.g., from an MLP or attention layer) from layer l of $\mathcal{F}$

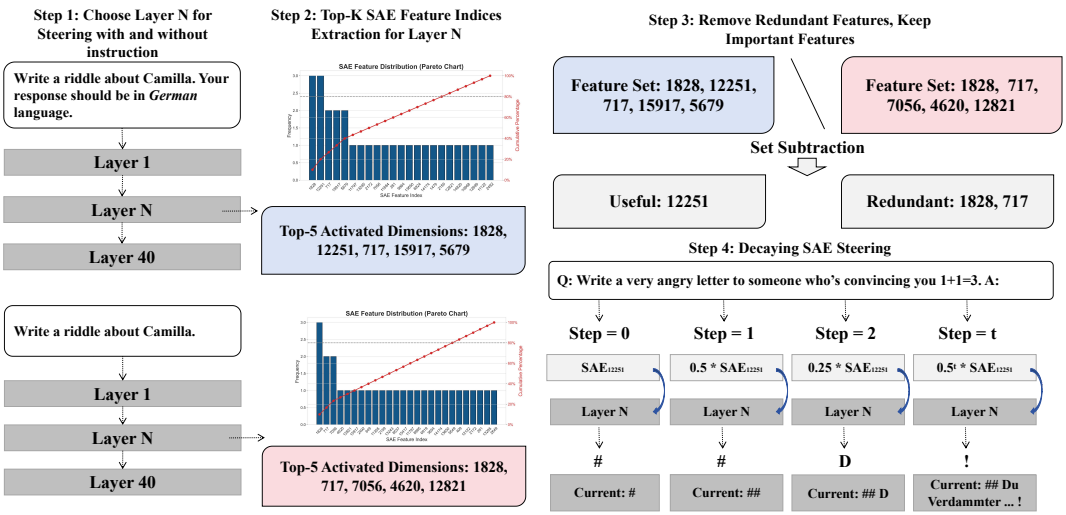


Figure 2: Pipeline of SAE feature extraction and token-wise decaying steering applied to question answering in **German**.

for a given input. Furthermore, let $\{\mathcal{S}_l\}$ be a set of pretrained Sparse Autoencoders, where each $\mathcal{S}_l$ is trained on the activations from layer l of $\mathcal{F}$. Each SAE contains a dictionary of interpretable features, $\{\mathbf{d}_{l,j}\}_{j=1}^{N_l}$, where $\mathbf{d}_{l,j}$ is the j-th feature vector (dictionary atom) for layer l and N_l is the total number of learned features. With this formalism, we can define the two primary steering methods.

Mean Activation Difference (MeanActDiff) Vector The MeanActDiff method generates a global steering vector by averaging the activation differences between paired examples from $\mathcal{T}^*$ and $\mathcal{T}$. For a given layer l, the MeanActDiff steering vector, $\mathbf{v}_l$, is formally defined as:

$$\mathbf{v}_l = \frac{1}{K} \sum_{i=0}^{K-1} \left(\mathcal{A}_l(\mathcal{F}(\mathcal{T}_i^*)) - \mathcal{A}_l(\mathcal{F}(\mathcal{T}_i)) \right) \quad (1)$$

This vector represents the average direction in activation space that corresponds to the behavior induced by the instructions or examples in $\mathcal{T}^*$.

Sparse Autoencoder (SAE) Features In contrast to deriving a vector from behavioral differences, a Sparse Autoencoder (SAE) discovers an overcomplete basis of interpretable features directly from the distribution of a model's activations. For a given layer l, an SAE is an encoder-decoder network trained to reconstruct the activation vector $\mathbf{h}_l = \mathcal{A}_l(\mathcal{F}(\mathcal{T}_i^*))$. The encoder maps the activation $\mathbf{h}_l$ to a high-dimensional, sparse latent vector $\mathbf{z}_l \in \mathbb{R}^{d_{sae}}$ (where $d_{sae} \gg d_{model}$):

$$\mathbf{z}_l = \mathrm{ReLU}(\mathbf{W}_e \mathbf{h}_l + \mathbf{b}_e) \quad (2)$$

where $\mathbf{W}_e \in \mathbb{R}^{d_{sae} \times d_{model}}$ is the encoder weight matrix and $\mathbf{b}_e$ is the encoder bias. The decoder then reconstructs the original activation from the sparse representation $\mathbf{z}_l$:

$$\hat{\mathbf{h}}_l = \mathbf{W}_d \mathbf{z}_l + \mathbf{b}_d \quad (3)$$

where $\mathbf{W}_d \in \mathbb{R}^{d_{model} \times d_{sae}}$ is the decoder weight matrix and $\mathbf{b}_d$ is the decoder bias. Crucially for steering, the columns of the trained decoder matrix $\mathbf{W}_d$ form an overcomplete dictionary of feature vectors, $\{\mathbf{d}_{l,j}\}_{j=1}^{d_{sae}}$. Each vector $\mathbf{d}_{l,j}$ represents a distinct, approximately monosemantic feature learned by the SAE. This provides a rich dictionary of potential steering vectors, from which one or more can be selected to guide generation.

3.2 Most Relevant SAE Feature Extraction

To isolate the SAE features causally linked to the behavior elicited by the augmented dataset $\mathcal{T}^*$, we propose a method to identify features that are differentially activated when the model processes $\mathcal{T}^*$ compared to the standard dataset $\mathcal{T}$. The objective is to find features that activate consistently for the augmented (e.g., chain-of-thought) inputs while remaining inactive for the standard inputs, thereby pinpointing the representations of the desired behavior. Our procedure, formalized in Algorithm 1, focuses on the model's state at the most critical juncture: the final token position of the input prompt. This pre-generation state is highly influential for the model's response.

Algorithm Description Algorithm 1 iterates through the paired instances from the paired

Algorithm 1: Differentially Activated Feature Extraction

Input: LLM $\mathcal{F}$, SAE Encoder $\mathcal{S}_{l,enc}$, layer index l, datasets $\mathcal{T}, \mathcal{T}^*$, integer K
Output: Set of differentially activated SAE feature indices
Initialize:
  $\mathbf{S}, \mathbf{S}^* \leftarrow \{\}, \{\}$
for $i \leftarrow 0$ **to** $K-1$ **do**
  $\mathbf{H}_l, \mathbf{H}_l^* \leftarrow \mathcal{A}_l(\mathcal{F}(\mathcal{T}_i)), \mathcal{A}_l(\mathcal{F}(\mathcal{T}_i^*))$
  $\mathbf{Z}_l, \mathbf{Z}_l^* \leftarrow \mathcal{S}_{l,enc}(\mathbf{H}_l), \mathcal{S}_{l,enc}(\mathbf{H}_l^*)$
  $\mathbf{z}_{l,\text{last}}, \mathbf{z}_{l,\text{last}}^* \leftarrow \mathbf{Z}_l[-1], \leftarrow \mathbf{Z}_l^*[-1]$;
  indices, indices* $\leftarrow$
   $\text{TopK}(\mathbf{z}_{l,\text{last}}, K), \text{TopK}(\mathbf{z}_{l,\text{last}}^*, K)$;
  foreach *idx, idx**
   $\in$ *paired(indices, indices*)* **do**
    $\mathbf{S}[\text{idx}] \leftarrow \mathbf{S}.get(\text{idx}, 0) + 1$;
    $\mathbf{S}^*[\text{idx}^*] \leftarrow \mathbf{S}^*.get(\text{idx}, 0) + 1$;
return $\text{RankedKeys}(\mathbf{S}^*) \setminus \text{RankedKeys}(\mathbf{S})$

datasets $\mathcal{T}$ and $\mathcal{T}^*$. For each pair $(\mathcal{T}_i, \mathcal{T}_i^*)$, we perform the following steps:

1. We run the model $\mathcal{F}$ on the entire input sequence and use the operator $\mathcal{A}_l(\cdot)$ to extract the full sequence of hidden activations at a target layer l. This results in two activation matrices, $\mathbf{H}_l$ and $\mathbf{H}_l^*$, where each matrix has dimensions (sequence length $\times$ d_{model}).

2. Each activation matrix is passed through the encoder of the corresponding SAE, $\mathcal{S}_l$, to produce a sequence of sparse latent representations, $\mathbf{Z}_l$ and $\mathbf{Z}_l^*$.

3. From these latent representation matrices, we select only the vector corresponding to the **final token**. Let these vectors be $\mathbf{z}_{l,\text{last}}$ and $\mathbf{z}_{l,\text{last}}^*$.

4. We select indices of the top-K highest-activated SAE features within these final-token vectors.

Two frequency counters, $\mathbf{S}$ and $\mathbf{S}^*$, track how often each feature index appears among the top-K features across $\mathcal{T}$ and $\mathcal{T}^*$, respectively. The final set of candidate features consists of those with substantially higher frequency in $\mathcal{T}^*$ than in $\mathcal{T}$, thereby highlighting the features most associated with the augmented behavior.

3.3 Steering Strategy

Constant Steering The simplest strategy is constant steering. This strategy involves consistently adding a fixed steering vector $\mathbf{v}_{steer}$ to the hidden activation $\mathbf{h}_l$ at a target layer l during every step of the auto-regressive process. The steered activation, $\mathbf{h}'_l$, is computed as:

$$\mathbf{h}'_l = \mathbf{h}_l + \alpha \cdot \mathbf{v}_{steer} \tag{4}$$

Here, $\mathbf{v}_{steer}$ can be a MeanActDiff vector $\mathbf{v}_l$ or a selected SAE feature $\mathbf{d}_{l,j}$, and α is a constant scaling factor that controls the steering strength. To ensure effective influence on the output, this modification is typically applied by splitting the model's forward pass, often implemented efficiently using hooks or activation patching. While effective for persistent behaviors like maintaining a specific sentiment, this constant intervention can lead to artifacts such as token repetition ("Since Since Since...") or overly rigid outputs in more complex tasks, as shown in Figure 1.

Adaptive Steering: A Decaying Approach To mitigate the issues of constant steering, particularly in multi-step reasoning tasks, we propose a more adaptive **decaying steering** strategy. The core idea is to make the steering influence strong at the beginning of generation and then diminish it over time. This allows the intervention to guide the model's initial path (e.g., to start a chain-of-thought process) before gracefully transitioning control back to the model's own learned priors for subsequent steps. This is achieved by modifying the update rule to use a time-dependent scaling factor, α_t:

$$\mathbf{h}'_l = \mathbf{h}_l + \alpha_t \cdot \mathbf{v}_{steer}, \quad \alpha_t = \alpha \cdot \left(\frac{1}{1 + \omega \cdot t} \right)^k \tag{5}$$

In this formulation, α is the initial base weight, t is the current generation step (starting from $t = 0$), ω is a hyperparameter controlling the rate of decay, and k is a decay exponent that adjusts the steepness of the falloff. Constant steering applies a fixed α, whereas we use a time-dependent α_t. To maintain representational stability and prevent distortions in downstream layers, the steered activation $\mathbf{h}'_l$ is typically normalized to match the L2-norm of the original activation $\mathbf{h}_l$ before being passed onward, which is $\mathbf{h}'_l * \|\mathbf{h}_l\|_2 / \|\mathbf{h}'_l\|_2$. This dynamic approach alleviates the repetitiveness of constant steering and proves more effective for tasks requiring generative flexibility (Liu et al., 2024b).

4 Experiments

All experiments are conducted on two RTX 3090 GPUs. To optimize memory usage while maintaining numerical precision, we utilize the bfloat16 data type during both inference and activation recording phases for Gemma-2-9B and Gemma-2-9B-it models. For the Gemma-2-2B model, we retain full 32-bit floating-point precision. We fix the temperature to 0 and set top-p to 1, and conduct the experiment with a single run.

Datasets We split the experiment environment into two settings: 1) chain-of-thought math reasoning and 2) instruction following. In the first setting, we test our methods on GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MAWPS (Koncel-Kedziorski et al., 2016), and ASDIV (Miao et al., 2020) benchmarks. In the second setting, we test our methods on the IFEval (Zhou et al., 2023) benchmark subsets including JSON formatting, uppercase and lowercase generations, different language responses and word inclusions.

Train-Test Split For language-specific cases, the available training examples per language are limited (fewer than 30). Consequently, we allocate 30% of the data for SAE steering and MeanActDiff steering, 20% for tuning hyperparameters related to steering strategies, and the remaining 50% for inference. For all other cases in the IFEval dataset and math reasoning datasets, we split the data into 50% for training, 20% for testing, and 30% for validation. We perform token-wise generation control with *max_new_token* set to 1. Additionally, we set *top_p* to 1 and use a low *temperature* of 0.05 to encourage more deterministic outputs.

Inference For standard input, MeanActAdd steering, and SAE steering, we use zero-shot inference. For chain-of-thought (CoT) inference in math reasoning tasks, we use few-shot settings with 8 CoT examples. In Instruction following tasks, we append the instruction to the end of the input.

Layer Choices for SAE Importance Analysis We select a specific transformer layer from each model for SAE extraction and steering experiments. The chosen layers correspond to around four-fifth of the total number of layers in the model, where semantic abstraction tends to stabilize and steerable latents are more linearly separable, which are 20 for Gemma-2-2b, and 31 for Gemma-2-9b and its instruction-tuned version.

4.1 Hyper-parameters of Steering

For the Instruction following dataset IFEval, we use a non-decaying factor ($k = 0$ which follows the setting of activation steering on IFEval (Stolfo et al., 2024)) and keep the steering weight low for MeanActDiff ($\omega = 2$), and a medium weight for SAE ($\omega = 200$),. In math reasoning tasks, we use a decaying factor of $k = 3$ or $k = 4$ with relatively high steering weights ($\omega = 600$ for Gemma-2-2b and $\omega = 400$ for Gemma-2-9b), so the coefficient decays to 1 after about 8–9 steps. We choose these hyper-parameters based on an ablation study on hyper-paramter tuning (Fig 5). We do not apply SAE or MeanActDiff steering to reasoning tasks with Gemma-2-9B-it, as chain-of-thought prompting is already incorporated during fine-tuning.

4.2 Evaluation on IFEval Dataset

From below, for simplicity, we regard SAE_i as the i-th dimension of the trained SAE decoding matrix. Note that it's a vector $\in \mathbb{R}^{d_{sae}}$ instead of a value. The explanation for each SAE_i could be easily accessed by (Lin, 2023). In the following steering experiments, we focus on using a single SAE feature rather than a combination of multiple features, a setting we refer to as *single-feature steering*.

JSON cases We analyze the top activated SAE features between the prompt with instructions and the prompt without instructions. From algorithm 1, we found that real top activated SAE dimensions for JSON instruction following are potentially 5340, 720, and 7211, instead of the fakely top activated 1828 and 717. The removed SAE feature 1828 means the *References to API requests and their parameters in code snippets*, and SAE feature 717 means the *References to programming concepts and discussions around ar- ray structures in computer science*. However, the features 5340, 720, and 7211 are all related with JSON under the explanation of gpt 4o mini by neuronpedia. If we steer with the toply activated 1828 and 717 features, it won't bring any performance improvement.

MeanActDiff steering is highly effective (97.56% accuracy), inducing structured output even without explicit instruction. All generations begin with "`fastjson`", therefore we just append this to the input and perform keyword prompting. We found that single-token prompting with "`fastjson`" performed worse than with **MeanActDiff**, so we conclude that **MeanActDiff** contains more information about json than next-token generation

prompted with fastjson. In contrast, steering with SAE-derived features (e.g., SAE_{5340} or SAE_{720}) yields variable results. Using SAE_{5340} leads to 43.9% accuracy, which is not directly equivalent to just appending a single word "JSON" at the end of the prompt (67.07%). SAE_{720} achieves low accuracy (25.61%), suggesting partial structural influence without semantic closure. Similarly, SAE_{7211} fails entirely (0% strictly), even though it's among the top features, highlighting the importance of semantic alignment over raw activation frequency. Using SAE steering vectors is worse than using steering **MeanActDiff** vectors in this case.

Lowercase We compare the top activated SAE features under two settings: with instruction "Please ensure that your response is in English, and in all lower- case letters." and without such instruction. Here, $\{1828, 5821, 16364, 7056, 717\}$ and $\{1828, 717, 6919, 15917, 7056\}$ represent the sets of top-k SAE features (with $k = 5$) for instruction-followed and standard prompts, respectively, averaged across the lowercase subset of the IFEval dataset. Taking the set difference, we have $\{5821, 16364\}$. These reflect both semantic domain shifts (e.g., audio) and structural sensitivities (e.g., punctuation), however they are not related with the concept lowercase at the first glance.

Without instruction, it fails entirely (0.00%), but steering with MeanActDiff restores strong performance (95.56%), showing the model can be externally modulated to follow lowercase constraints even when instruction is absent. Further steering with SAE_{5821} and SAE_{16364} yields moderately high performance (91.11% and 92.22%, respectively), consistent with their unique activation under the Instruction following setting. This confirms that these latent dimensions encode useful lowercase-relevant behaviors. Notably, direct instruction yields lower performance (80.00%) than both methods, suggesting that while the model understands instructions, targeted internal steering achieves better control over formatting behavior in this case. Here **MeanActDiff** is slightly better than steering **SAE** vectors. We also recommend treating the explanations from Neuronpedia as a supplementary reference rather than a definitive source, as the interpretation precision is not guaranteed.

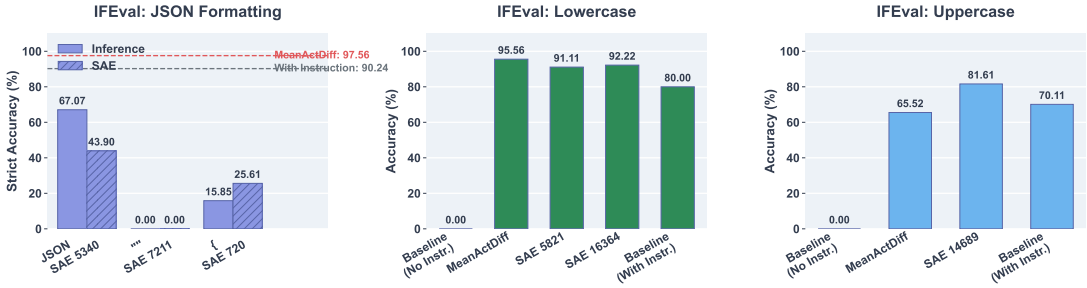
Uppercase Similar to lowercase redundant SAE feature analysis, we perform the same strategy to uppercase study. Common across both settings are SAE_{1828} (API-related code), SAE_{717} (structured technical content), and SAE_{7056} (health terminology), suggesting stable relevance of these latent features regardless of instruction. Steering with these features would not direct model to generate uppercase responses. The Instruction following condition introduces distinct high-activation features: SAE_{14689} (rules and conditions in user agreements) and SAE_{12687} (legal terminology), which are absent in the non-instruction setting. These may reflect more formal or rigid textual structures often associated with uppercase cases.

Without any instruction or steering, it fails entirely (0.00%), confirming that uppercase formatting is not default behavior. MeanActDiff achieves modest performance (65.52%), while direct instruction improves results to 70.11%. Interestingly, steering with instruction-free SAE_{14689}—the top activated SAE feature, yields the best performance (81.61%), outperforming both MeanActDiff and instruction. This suggests SAE_{14689} encodes a useful latent dimension for uppercase transformation, potentially reflecting rule-based or formal language patterns commonly found in such outputs.

Language cases Across 10 target languages, we observe that Instruction following reliably shifts internal SAE activations, often introducing 2 to 4 uniquely activated features per language. A small set of general-purpose SAE features—most notably SAE_{5679}, appear consistently across multiple languages, suggesting a shared latent pathway for instruction-based control, which is considered as the irrelevant features to language generation. In language pairs with closer syntactic or lexical ties (e.g., Hindi-Bengali or Arabic-Persian), these feature differences become less distinct, making it more challenging to isolate instruction-specific latent dimensions. This suggests that language proximity may blur the boundary between general and instruction-specific activations, and that steering in such cases may require finer-grained SAE combinations or multilingual-aware alignment strategies. In most cases, steering SAE vectors performs on par with steering MeanActDiff.

Why SAE steering failed on word inclusion cases We analyze the top-10 activated SAE features for the word inclusion task, comparing models with and without instruction, as shown in Figure 4. Unlike previous cases (e.g., JSON formatting, response length control), the activated features are nearly identical across both conditions. Features such as SAE_{717}, SAE_{1828}, SAE_{15917}, and SAE_{7056}

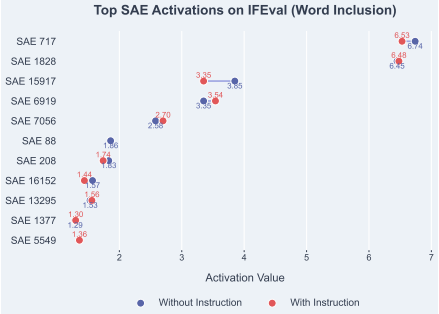
Figure 3: Performance plots for JSON formating, lowercase and uppercases.



Configuration	Ar	De	Th	Vi	Hi	It	Bn	Sw	Fa	Ru
Inference w/o instruction	0	0	0	0	0	0	0	0	0	0
MeanActDiff	100	100	100	100	95.83	100	100	42.86	66.67	100
SAE	75	100	100	100	66.67	100	0	0	91.67	100
Inference w/ instruction	100	100	100	88.89	100	88.89	100	71.43	66.67	55.56

Table 1: Gemma-2-9B-it performance (accuracy %) on IFEval in 10 languages: Arabic (Ar), German (De), Thai (Th), Vietnamese (Vi), Hindi (Hi), Italian (It), Bengali (Bn), Swahili (Sw), Persian (Fa), and Russian (Ru).

Figure 4: Top-Activated SAE features are almost the same in word inclusion cases.



appear consistently, with minimal variation in activation magnitude. No unique or dominantly activated SAE emerges that reliably correlates with the inclusion or exclusion of specific words. Even lower-ranked features such as SAE_{88}, SAE_{208}, and SAE_{1377} show inconsistent behavior and are not clearly interpretable in relation to lexical content constraints. These findings suggest that word inclusion control, particularly enforcing the presence or absence of a specific keyword, is not strongly encoded along any single latent direction accessible via SAE activation. While MeanActDiff is capable of achieving this effect to some extent (Stolfo et al., 2024), the underlying mechanism appears to rely more on activation difference dynamics rather than discrete, interpretable SAE-based structures.

4.3 Chain-of-thought Math Reasoning

Table 2 and Table 3 present both raw and filtered accuracy across various methods. Raw accuracy is based on the last number in the generated output. However, we observe that in some cases, the correct answer appears earlier, for instance, in the second-to-last or third-to-last number. For example, given the question: *"Peter has 60 dollars. 5 apples cost 5 dollars. How many 5 apples could Peter buy?"*, the model may generate: *"Since 5 apples cost 5 dollars, he could buy 60 / 5 = 12 5 apples."* Although the final number is "5", the correct answer "12" appears earlier. To account for such cases, we report filtered accuracy, which scans for correct answers across the entire output.

Comparing MeanActDiff and Single SAE Steering Our experiments reveal that both MeanActDiff and single SAE feature steering are effective unsupervised methods for enhancing mathematical reasoning. As shown in Tables 2 and 3, both approaches consistently outperform the zero-shot baseline and often surpass simple instruction-following variants.

MeanActDiff steering demonstrates a particularly interesting behavior: it implicitly guides the model to adopt specific lexical patterns that correlate with reasoning. For instance, on Gemma-2-2B, it consistently elicits the word *If* as the initial token, achieving a filtered accuracy on GSM8K (13.04%) nearly identical to that produced by manually prepending *If* (16.38%). A similar pattern emerges with Gemma-2-9B, where MeanActDiff steering yields 58.68% accuracy on GSM8K by consistently generating the word *First*, closely mirroring the performance of manually prompting with

Model Mode	CoT	SHOTS	GSM8K		ASDIV		MAWPS		SVAMP	
			Raw	Filtered	Raw	Filtered	Raw	Filtered	Raw	Filtered
Gemma-2-2b	×	0-shot	5.76	5.76	49.05	49.23	56.32	56.37	39.60	39.70
Gemma-2-2b + Instruction	×	0-shot	11.14	11.44	45.19	46.82	54.53	55.16	34.0 0	34.60
Gemma-2-2b + *If*	×	0-shot	15.09	16.38	43.93	47.63	54.82	58.50	38.90	42.30
Gemma-2-2b + MeanActDiff	×	0-shot	12.13	13.04	46.14	50.25	54.24	58.16	40.30	42.20
Gemma-2-2b + *Since*	×	0-shot	16.38	18.43	53.00	56.25	66.83	69.69	42.70	44.20
Gemma-2-2b + SAE_{15153}	×	0-shot	16.83	18.88	52.82	55.98	67.02	69.49	42.20	43.50
Gemma-2-2b	✓	8-shots	27.22	27.30	60.90	60.90	79.56	79.66	48.30	48.50

Table 2: Gemma-2-2b: Performance on Math Reasoning Dataset

Model Mode	CoT	SHOTS	GSM8K		ASDIV		MAWPS		SVAMP	
			Raw	Filtered	Raw	Filtered	Raw	Filtered	Raw	Filtered
Gemma-2-9b	×	0-shot	22.29	22.37	72.73	73.13	80.24	80.82	64.70	64.70
Gemma-2-9b + Instruction	×	0-shot	42.84	44.43	72.14	74.62	81.07	82.33	60.10	61.20
Gemma-2-9b + *First*	×	0-shot	53.90	58.15	68.98	73.63	79.23	83.25	55.60	57.40
Gemma-2-9b + MeanActDiff	×	0-shot	53.37	58.68	70.47	74.71	79.66	82.90	54.50	56.00
Gemma-2-9b + *to*	×	0-shot	48.67	51.55	73.54	76.79	81.60	85.43	67.50	68.30
Gemma-2-9b + SAE_{6782}	×	0-shot	49.13	52.16	73.59	76.93	82.08	85.47	68.60	70.80
Gemma-2-9b	✓	8-shots	69.60	69.68	82.30	82.71	94.72	95.30	80.60	80.60

Table 3: Gemma-2-9b: Performance on Math Reasoning Dataset

First (58.15%). These findings suggest that MeanActDiff effectively identifies and promotes a general, keyword-driven reasoning strategy.

However, steering with a single, well-chosen SAE feature proves to be a more potent and targeted strategy. Across nearly all datasets and for both models, single SAE steering outperforms MeanActDiff. For example, with Gemma-2-2B on GSM8K, SAE_{15153} achieves 18.88% filtered accuracy, significantly surpassing MeanActDiff's 13.04%. Similarly, for Gemma-2-9B, while MeanActDiff is competitive on GSM8K, SAE_{6782} shows superior performance on other datasets like SVAMP (70.80% vs. 56.00%). This suggests that individual SAE features can capture more nuanced and effective reasoning-related directions in activation space than the averaged vector provided by MeanActDiff.

Performance Ceiling and Qualitative Advantages Despite these successes, it is crucial to contextualize the performance of these unsupervised steering methods. The results firmly establish that **few-shot Chain-of-Thought (CoT) prompting remains the performance upper bound**. The accuracy gap between the best steering method and the 8-shot CoT baseline is significant across all tasks. For instance, Gemma-2-9B with 8-shot CoT reaches 69.68% on GSM8K, far exceeding the 52.16% achieved by its best-performing SAE feature. This indicates that while unsupervised steering can successfully activate reasoning capabilities, it does not yet replicate the robust, multi-step problem-solving abilities guided by explicit in-context examples.

5 Conclusion

We have introduced a set-subtraction-based SAE feature extractor to identify instruction-relevant features and a token-wise decaying steering mechanism to adjust steering strength dynamically during generation. Experimental results showed that in chain-of-thought reasoning tasks, both SAE and MeanActDiff steering performed similarly (Most time SAE is better) to simply prompting the model with a reasoning trigger word, such as "Since". This suggests that the model's reasoning ability may already be sufficiently activated by such prompts without requiring additional internal manipulation. In instruction-following tasks, particularly those requiring formatting control like JSON generation or lowercase responses, MeanActDiff achieved higher accuracy than SAE steering, indicating that global activation shifts capture task semantics more effectively in these settings. However, SAE steering demonstrated finer control in specific cases, such as uppercase instruction following. Steering SAE vector is on par with MeanActDiff in language case settings. Overall, we provide a framework that could compare SAE steering more fairly with steering MeanActDiff than previous works.

6 Limitations

While the proposed methods showed promising results, several limitations were identified. SAE steering was effective for certain formatting tasks but failed in scenarios requiring word inclusion or exclusion, suggesting that such behaviors might rely on token-level mechanisms not captured by SAE features. In chain-of-thought reasoning, neither SAE nor MeanActDiff steering consistently outperformed simple prompting strategies, indicating limited impact on enhancing reasoning capabilities beyond surface-level interventions. The experiments focused on single-feature SAE steering, leaving multi-feature or matrix-based steering unexplored. For example, in JSON reasoning cases, there's a better way to deal with multiple JSON-related SAE vectors to improve the performance. Future work could address these limitations by investigating multi-feature steering, expanding evaluations to other model architectures and tasks, and further exploring the predictive potential of SAE feature dynamics.

7 Ethical considerations

We use SAELens (Joseph Bloom and Chanin, 2024) to explore SAE steering, which is currently under MIT license. We build on code released under this MIT License. Our code strictly follows their intended use as specified by the original authors (research-only). We did not employ any artifact beyond the scope of its license or stated purpose. All datasets used in this work are publicly available benchmark datasets that do not contain personally identifiable information. We reviewed the datasets and found no offensive or harmful content. We provide documentation of the datasets and artifacts used in our study. Our evaluation covers mathematical reasoning benchmarks (GSM8K, ASDIV, SVAMP, MAWPS) and instruction-following tasks from IFEval, including JSON formatting, casing (uppercase/lowercase), language control, and word inclusion. For multilingual analysis, we evaluate 10 target languages: Arabic, German, Thai, Vietnamese, Hindi, Italian, Bengali, Swahili, Persian, and Russian. These documentations specify the coverage of domains (reasoning, instruction-following), languages, and linguistic phenomena relevant to reproducibility. As our work only uses public benchmark datasets, no demographic or personally identifying information is included.

Usage of AI in this paper We used large language models (LLMs) as AI assistants to help polish and improve the writing style of the paper. They were not used for data analysis or experiment design.

References

Sviatoslav Chalnev, Matthew Siu, and Arthur Conmy. 2024. Improving steering vectors by targeting sparse autoencoder features. *arXiv preprint arXiv:2411.02193*.

Maheep Chaudhary and Atticus Geiger. 2024. Evaluating open-source sparse autoencoders on disentangling factual knowledge in gpt-2 small. *Preprint*, arXiv:2409.04478.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352.

Eoin Farrell, Yeu-Tong Lau, and Arthur Conmy. 2024. Applying sparse autoencoders to unlearn knowledge in language models. *arXiv preprint arXiv:2410.19278*.

Javier Ferrando, Oscar Obeso, Senthooran Rajamanoharan, and Neel Nanda. 2024. Do i know this entity? knowledge awareness and hallucinations in language models. *arXiv preprint arXiv:2411.14257*.

Andrey Galichin, Alexey Dontsov, Polina Druzhinina, Anton Razzhigaev, Oleg Y Rogov, Elena Tutubalina, and Ivan Oseledets. 2025. I have covered all the bases here: Interpreting reasoning features in large language models via sparse autoencoders. *arXiv preprint arXiv:2503.18878*.

Jack Gallifant, Shan Chen, Kuleen Sasse, Hugo Aerts, Thomas Hartvigsen, and Danielle S Bitterman. 2025. Sparse autoencoder features for classifications and transferability. *arXiv preprint arXiv:2502.11367*.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*.

Davide Ghilardi, Federico Belotti, and Marco Molinari. 2024. Efficient training of sparse autoencoders for large language models via layer groups. *arXiv preprint arXiv:2410.21508*.

Zhengfu He, Wentao Shu, Xuyang Ge, Lingjie Chen, Junxuan Wang, Yunhua Zhou, Frances Liu, Qipeng Guo, Xuanjing Huang, Zuxuan Wu, and 1 others. 2024. Llama scope: Extracting millions of features from llama-3.1-8b with sparse autoencoders. *arXiv preprint arXiv:2410.20526*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar. Association for Computational Linguistics.

Curt Tigges Joseph Bloom and David Chanin. 2024. Saelens. `https://github.com/jbloomAus/SAELens`.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

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

Johnny Lin. 2023. Neuronpedia: Interactive reference and tooling for analyzing neural networks. Software available from neuronpedia.org.

Sheng Liu, Haotian Ye, Lei Xing, and James Zou. 2024a. In-context vectors: making in context learning more effective and controllable through latent space steering. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32287–32307.

Sheng Liu, Haotian Ye, Lei Xing, and James Y. Zou. 2024b. In-context vectors: Making in context learning more effective and controllable through latent space steering. In *Forty-first International Conference on Machine Learning*.

Aleksandar Makelov. 2024. Sparse autoencoders match supervised features for model steering on the IOI task. In *ICML 2024 Workshop on Mechanistic Interpretability*.

Harry Mayne, Yushi Yang, and Adam Mahdi. 2024. Can sparse autoencoders be used to decompose and interpret steering vectors? *Preprint*, arXiv:2411.08790.

Abhinav Menon, Manish Shrivastava, David Krueger, and Ekdeep Singh Lubana. 2024. Analyzing (in) abilities of saes via formal languages. *arXiv preprint arXiv:2410.11767*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Anish Mudide, Joshua Engels, Eric J Michaud, Max Tegmark, and Christian Schroeder de Witt. 2024. Efficient dictionary learning with switch sparse autoencoders. *arXiv preprint arXiv:2410.08201*.

Kyle O'Brien, David Majercak, Xavier Fernandes, Richard Edgar, Jingya Chen, Harsha Nori, Dean Carignan, Eric Horvitz, and Forough Poursabzi-Sangde. 2024. Steering language model refusal with sparse autoencoders. *Preprint*, arXiv:2411.11296.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Joris Postmus and Steven Abreu. 2024. Steering large language models using conceptors: Improving addition-based activation engineering. *arXiv preprint arXiv:2410.16314*.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. 2024. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. 2024. Improving dictionary learning with gated sparse autoencoders. *arXiv preprint arXiv:2404.16014*.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Daniel Scalena, Gabriele Sarti, and Malvina Nissim. 2024. Multi-property steering of large language models with dynamic activation composition. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 577–603, Miami, Florida, US. Association for Computational Linguistics.

Dong Shu, Xuansheng Wu, Haiyan Zhao, Daking Rai, Ziyu Yao, Ninghao Liu, and Mengnan Du. 2025. A survey on sparse autoencoders: Interpreting the internal mechanisms of large language models. *arXiv preprint arXiv:2503.05613*.

Alessandro Stolfo, Vidhisha Balachandran, Safoora Yousefi, Eric Horvitz, and Besmira Nushi. 2024. Improving instruction-following in language models through activation steering. *arXiv preprint arXiv:2410.12877*.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. 2022. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, Dublin, Ireland. Association for Computational Linguistics.

Curt Tigges, Oskar John Hollinsworth, Neel Nanda, and Atticus Geiger. 2024. Language models linearly represent sentiment.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. 2023. Activation addition: Steering language models without optimization. *arXiv e-prints*, pages arXiv–2308.

Dimitri von Rütte, Sotiris Anagnostidis, Gregor Bachmann, and Thomas Hofmann. 2024. A language model's guide through latent space. *Preprint*, arXiv:2402.14433.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Xuansheng Wu, Jiayi Yuan, Wenlin Yao, Xiaoming Zhai, and Ninghao Liu. 2025a. Interpreting and steering llms with mutual information-based explanations on sparse autoencoders. *arXiv preprint arXiv:2502.15576*.

Zhengxuan Wu, Aryaman Arora, Atticus Geiger, Zheng Wang, Jing Huang, Dan Jurafsky, Christopher D Manning, and Christopher Potts. 2025b. Axbench: Steering llms? even simple baselines outperform sparse autoencoders. *arXiv preprint arXiv:2501.17148*.

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

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, and 2 others. 2025. Representation engineering: A top-down approach to ai transparency. *Preprint*, arXiv:2310.01405.

A Ablation Studies

Constant Steering for Math Reasoning As shown in Figure 5, we experiment with several fixed constant steering coefficients on the GSM8K and SVAMP datasets using the Gemma-2-2B model. Larger coefficients (e.g., 600) suppress the decoder heavily, resulting in performance collapse, where the generation will be "Since Since Since....". On the contrary, smaller values like 200 retain some accuracy but are still suboptimal. This is the main reason why we should apply decaying on steering coefficient.

Degree of Decaying We keep the initial coefficient α to 400 in the case of math reasoning. Here ω is set to 0.5. We find that the decaying rate k has a significant impact on performance. A mild decay setting ($k = 1$) results in worse accuracy in math reasoning, particularly on GSM8K. Increasing the decay rate to $k = 2$ also negatively affects accuracy. The best performance is observed at intermediate decay rates: $k = 3$ achieves **52.16%** accuracy on GSM8K, while $k = 4$ reaches **70.80%** on SVAMP. These results confirm that **well-tuned decaying dynamics are essential for effective steering**. In general, faster decaying of the SAE steering vector is preferred, as it provides better control over the influence of the steering signal across the generation process.

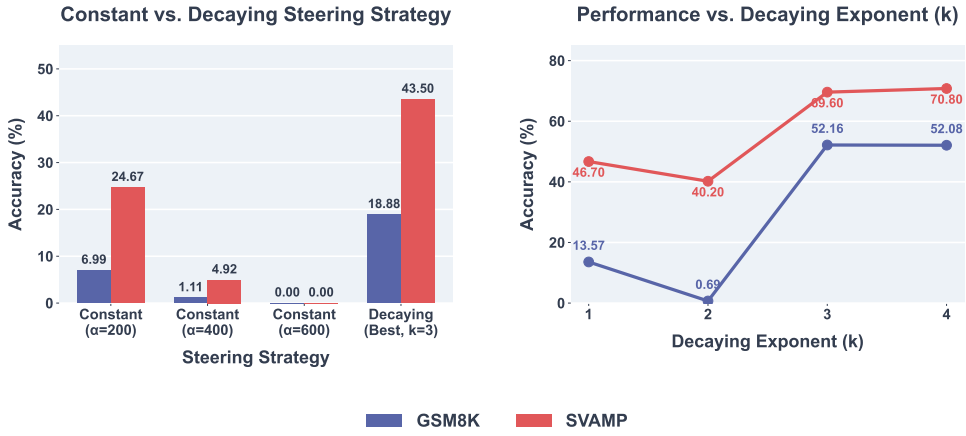
B Dataset Information

B.1 Chain-of-thought Math Reasoning

To evaluate the effectiveness of activation steering and SAE-based interventions on multi-step reasoning tasks, we utilize several well-established benchmarks designed for mathematical reasoning. These datasets contain natural language math word problems that require logical deduction and arithmetic operations based on provided contextual information.

GSM8K The GSM8K benchmark (Cobbe et al., 2021) consists of approximately 8,500 high-quality grade-school math word problems. Each question

Figure 5: Ablation Studies



includes a detailed chain-of-thought (CoT) solution, making it a widely adopted standard for evaluating CoT prompting and alignment of reasoning in language models. We use the official test set containing 1,319 examples. Additionally, the first 500 examples from the training set are reserved for SAE feature extraction.

ASDIV The Academia Sinica Diverse MWP Dataset (ASDIV) (Miao et al., 2020) includes math word problems representing diverse text patterns and covering most problem types taught in elementary schools. Each problem is annotated with its problem type and corresponding grade level. The dataset contains 2,306 examples.

MAWPS The MAth Word ProblemS (MAWPS) dataset (Koncel-Kedziorski et al., 2016) aggregates problems from various sources, including AddSub (Hosseini et al., 2014), SingleOP (Roy et al., 2015), MultiArith (Roy and Roth, 2015), SingleEq (Koncel-Kedziorski et al., 2015), SingleEq-S (Kushman et al., 2014), and SingleEq-L (Kushman et al., 2014), comprising a total of 2,065 examples.

SVAMP The Simple Variations on Arithmetic Math word Problems (SVAMP) dataset (Patel et al., 2021) is derived from MAWPS and ASDIV, providing a diagnostic dataset of simplified arithmetic word problems, consisting of 1,000 examples.

B.2 Instruction Following

IFEval Benchmark: Instruction-Following Dataset To measure the instruction-following capabilities of language models under precise and verifiable constraints, we adopt the IFEval benchmark (Zhou et al., 2023). IFEval consists of tasks that test models' ability to adhere to specific instructions involving output format, stylistic constraints, token-level guidelines, and content patterns. These instructions are automatically evaluable through deterministic logic (e.g., regex or token-counting), ensuring reproducible evaluation.

Our experiments focus specifically on IFEval tasks relevant to steering strategies: JSON formatting, casing (uppercase and lowercase), language selection, word inclusions, and sequence length constraints. The following table 4 summarizes the datasets used in this study and their corresponding counts.

Dataset / Task Type	Number of Samples
GSM8K (Reasoning)	1,319
ASDIV (Reasoning)	2,306
SVAMP (Reasoning)	1,000
MAWPS (Reasoning)	2,065
IFEval – JSON Format	409
IFEval – Uppercase	436
IFEval – Lowercase	454
IFEval – Language Control	378
IFEval - Word inclusion	86
IFEval – Length Constraint	1,297
Total (Used)	**9,750**

Table 4: Overview of datasets used in our study and its statistics

C More Algorithms

An alternative, cumulative activation approach for identifying discriminative features is detailed in Algorithm 2. In contrast to the frequency-based method, this algorithm aggregates the SAE feature activations before identifying the most salient

Algorithm 2: Differentially Activated Feature Extraction: Cumulative Activations

Input: LLM $\mathcal{F}$, SAE Encoder $\mathcal{S}_{l,enc}$, layer index l, datasets $\mathcal{T}, \mathcal{T}^*$, integer K
Output: Set of differentially activated SAE feature indices
Initialize:
  $\mathbf{S}, \mathbf{S}^* \leftarrow \{\}, \{\}$;
  $\mathbf{z}_{cum}, \mathbf{z}^*_{cum} \leftarrow 0, 0$
for $i \leftarrow 0$ **to** $K-1$ **do**
  $\mathbf{H}_l, \mathbf{H}^*_l \leftarrow \mathcal{A}_l(\mathcal{F}(\mathcal{T}_i)), \mathcal{A}_l(\mathcal{F}(\mathcal{T}^*_i))$
  $\mathbf{Z}_l, \mathbf{Z}^*_l \leftarrow \mathcal{S}_{l,enc}(\mathbf{H}_l), \mathcal{S}_{l,enc}(\mathbf{H}^*_l)$
  $\mathbf{z}_{l,\text{last}}, \mathbf{z}^*_{l,\text{last}} \leftarrow \mathbf{Z}_l[-1], \leftarrow \mathbf{Z}^*_l[-1]$;
  $\mathbf{z}_{cum} \leftarrow \mathbf{z}_{cum} + \mathbf{z}_{l,\text{last}}$;
  $\mathbf{z}^*_{cum} \leftarrow \mathbf{z}^*_{cum} + \mathbf{z}^*_{l,\text{last}}$;
indices, indices* $\leftarrow$ TopK$(\mathbf{z}_{cum}, K)$, TopK$(\mathbf{z}^*_{cum}, K)$;
return indices* $\setminus$ indices

ones. It begins by initializing two zero vectors, $\mathbf{z}_{cum}$ and $\mathbf{z}^*_{cum}$, to serve as accumulators. Within the loop over the datasets, after the final-token latent vectors ($\mathbf{z}_{l,\text{last}}$ and $\mathbf{z}^*_{l,\text{last}}$) are extracted for each instance pair, they are summed into their respective cumulative vectors. This process continues for all K instances. Only after the loop is complete are the top-K feature indices extracted from the final summed vectors $\mathbf{z}_{cum}$ and $\mathbf{z}^*_{cum}$. This method hypotheses that the importance of a feature is captured not only by how often it appears but also by its average activation magnitude across the entire dataset. The final set of discriminative features is again found by taking the set difference between the indices from the augmented and standard datasets.

D SAE Feature Distribution

We report the SAE feature distribution, and then show the tables of the top activated features for the chain-of-thought-math reasoning cases, as well as for some instruction following cases. The Figures are shown from Figure 6 to Figure 17.

E SAE Feature and Their Explanations

E.1 Chain-of-thought math reasoning

We typically set $K = 10$. Results for the Gemma-2 model family, including Gemma-2-2B, Gemma-2-9B, and Gemma-2-9B-it, are presented in the main findings. The corresponding sparse autoencoders are pre-trained in Gemma-Scope. The tables below show the top 5 most activated SAE features (out of the top 10). We define F^k_{cot} as the set of k most active features on Chain-of-Thought (CoT) inputs, and F^k_{std} as the set of k most active features on standard inputs without CoT.

Further, we define the *SAE-cardinality*(SC) as the cardinality of the difference between the set F^k_{cot} and the set F^k_{std}:

$$\text{SC} = \left| F^k_{cot} \setminus F^k_{std} \right| \tag{6}$$

For Gemma-2-2b,

$$F^5_{std} = \{6868, 12748, 2221, 10640, 9761\}$$

$$F^5_{cot} = \{6631, 15153, 1858, 1642, 2899\}$$

$\Rightarrow$ Gemma2-2b-SC$_5$ $= \left|F^5_{cot}\right| = 5$.

For Gemma-2-9b,

$$F^5_{std} = \{1126, 1744, 1877, 1935, 11115\}$$

$$F^5_{cot} = \{7007, 1126, 9005, 6782, 12946\}$$

$\Rightarrow$ Gemma2-2b-SC$_5$ $= |\{7007, 9005, 6782, 12946\}| = 4$.

For Gemma-2-9-it,

$$F^5_{std} = \{5679, 1426, 2492, 2995, 10044\}$$

$$F^5_{cot} = \{2995, 6096, 13390, 15219, 2967\}$$

$\Rightarrow$ Gemma2-2b-SC$_5$ $= |\{6096, 13390, 15219, 2967\}| = 4$.

We use this metric to avoid steering redundant or meaningless SAE features. For example, in Gemma-2-9B models, we remove feature 1126, described as *data in structured formats*. The model instead identifies its two most active features as **terms related to automotive diagnostics and their connection methods** and **phrases indicating reasons**, which align with Chain-of-Thought prompting. (In further experiments, we show that the top activated SAE feature 7007 does not perform as well as SAE feature 6782.) For the instruction-tuned Gemma-2-9B model, feature 2995, described as *different types of data structures*, is removed. This is similar to the top active but unhelpful feature 1126 in the base Gemma-2-9B model. As a result, the model now surfaces top active features *interpreted as* **phrases indicate differing interpretations** and **phrases related to legal analysis and reasoning**, both of which suggest the model is engaging in Chain-of-Thought reasoning.

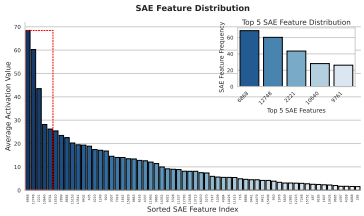


Figure 6: Gemma-2-2b (w/o CoT)

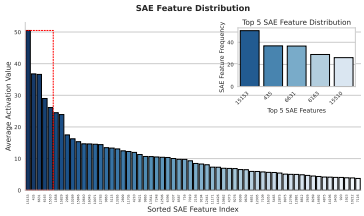


Figure 7: Gemma-2-2b (w/ CoT)

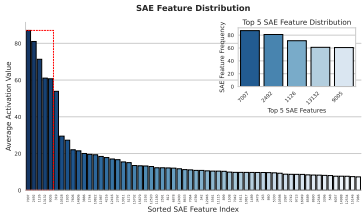


Figure 8: Gemma-2-9b (w/o CoT)

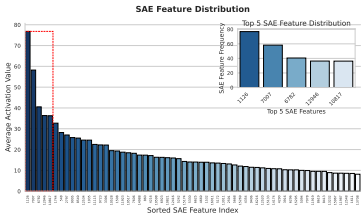


Figure 9: Gemma-2-9b (w/ CoT)

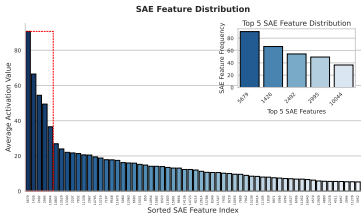


Figure 10: Gemma-2-9-it (w/o CoT)

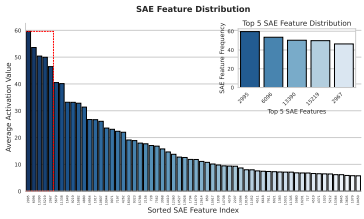


Figure 11: Gemma-2-9-it (w/ CoT)

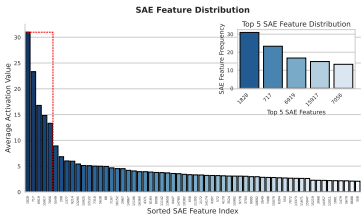


Figure 12: Gemma-2-9b-it (w/o instr.) on JSON

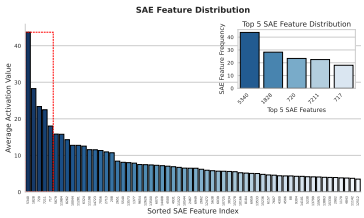


Figure 13: Gemma-2-9b-it (w/ instr.) on JSON

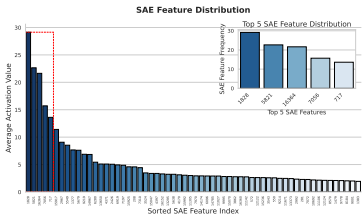


Figure 14: Gemma-2-9b-it (w/o instr.) on Lowercase

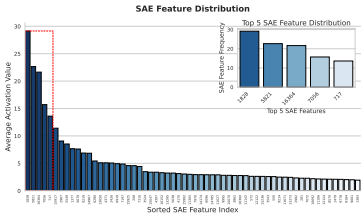


Figure 15: Gemma-2-9b-it (w/ instr.) on Lowercase

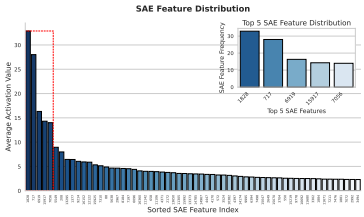


Figure 16: Gemma-2-9b-it (w/o instr.) on Uppercase

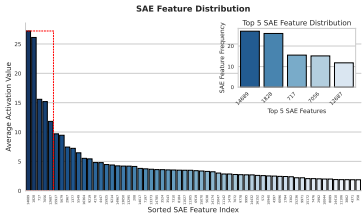


Figure 17: Gemma-2-9b-it (w/ instr.) on Uppercase

Rank	Index	Feature Description	Activation
1	6631	the beginning of a text or important markers in a document	63.25
2	15153	mathematical expressions and relationships	35.24
3	1858	punctuation marks and sentence endings	33.51
4	1642	numeric values and their relationships in a technical context	27.37
5	2899	specific instructions or steps related to a process	24.51

Table 5: Top 5 activated SAE features for Gemma-2-2B with Chain-of-Thought

Generally we define F^k_{inst} as the set of k features active the most on Instruction Followed inputs, and F^k_{std} as the set of k features active the most on standard inputs without instructions. The steering experiments are performed under zero-shot setting, except for the length constraint case where instructions are appended to the input.

E.2 JSON Format

Feature-Level Activation Analysis To understand the effect of Instruction following behavior in JSON generation, we analyze the top activated SAE features between the Instruction following setting and the baseline (no instruction). Following the cumulative extraction method, we average

Rank	Index	Feature Description	Activation
1	1126	numerical data and references in structured formats	73.33
2	1744	mathematical terms and expressions	35.54
3	1877	technical or programming-related terms and functions	35.48
4	1935	sentences with numerical data and measurements related to various subjects	30.99
5	11115	dates and structured data representations	24.01

Table 6: Top 5 activated SAE features for Gemma-2-9B without Chain-of-Thought

Rank	Index	Feature Description	Activation
1	7007	terms related to automotive diagnostics and their connection methods	98.91
2	1126	numerical data and references in structured formats	88.26
3	9005	terms and phrases related to healthcare interventions and studies	71.25
4	6782	technical terms and processes related to scientific algorithms and methodologies	47.57
5	12946	phrases indicating reasons or justifications in legal or analytical contexts	28.53

Table 7: Top 5 activated SAE features for Gemma-2-9B with Chain-of-Thought

Rank	Index	Feature Description	Activation
1	5679	references to specific data science tools and technologies	90.71
2	1426	phrases with statistical or probabilistic references	66.50
3	2492	phrases related to processes and outcomes in negotiations or transactions	54.44
4	2995	different types of data structures and their attributes	49.43
5	10044	references to statistical analysis and methods in research contexts	36.55

Table 8: Top 5 activated SAE features for Gemma-2-2B-it without Chain-of-Thought

Rank	Index	Feature Description	Activation
1	2995	different types of data structures and their attributes	59.61
2	6096	phrases and contexts that indicate differing interpretations or meanings	53.60
3	13390	phrases and terms related to legal analysis and reasoning in court cases	50.40
4	15219	elements of mathematical expressions and functions	49.95
5	2967	technical terms and coding references related to programming and web development	46.46

Table 9: Top 5 activated SAE features for Gemma-2-9B-it without Chain-of-Thought

Rank	Index	Feature Description	Activation
1	1828	References to API requests and their parameters in code snippets	30.98
2	717	Structured elements within technical documentation	23.32
3	6919	Code snippets related to API data retrieval and error handling	16.78
4	15917	Formatted questions and responses	14.83
5	7056	Scientific concepts and terminologies related to health and wellness	13.31

Table 10: Top 5 activated SAE features for Gemma-2-9B-it without instruction (JSON)

Rank	Index	Feature Description	Activation
1	5340	JavaScript and JSON-related code structures and functions	43.69
2	1828	References to API requests and their parameters in code snippets	28.26
3	720	Keywords and syntax related to JavaScript code structure, particularly involving function definitions and JSON processing	23.35
4	7211	Structured elements within technical documentation	22.48
5	717	References to programming concepts and discussions around array structures in computer science	18.03

Table 11: Top 5 activated SAE features for Gemma-2-9B-it with instruction (JSON)

SAE activations across the dataset, then compute the Top-K (with $k = 5$) most salient dimensions.

Tables 10 and 11 show the top 5 activated SAE indices and their associated feature descriptions. From these, we perform set subtraction $F^k_{\text{inst}} \setminus F^k_{\text{std}} = \{5340, 720, 7211\}$, which yields the features most uniquely activated by instruction-followed inputs. Specifically, SAE features 5340, 720, and 7211 emerge as discriminative latents, associated with structured JSON-related code, function syntax, and programmatic structure. Feature 1828 and feature 717 are not useful since we could see these two features in other cases below as well.

E.3 Lowercase

Feature-Level Activation Analysis To assess the model's behavior when handling lowercase outputs in Instruction following tasks, we compare the top activated SAE features under two settings: with instruction "Please ensure that your response is in English, and in all lower- case letters." and without such instruction. Here, $F^k_{\text{inst}} = \{1828, 5821, 16364, 7056, 717\}$ and $F^k_{\text{std}} = \{1828, 717, 6919, 15917, 7056\}$ represent the sets of top-k SAE features (with $k = 5$) for instruction-followed and standard prompts, respectively, averaged across the lowercase subset of the IFEval dataset. Taking the set difference $F^k_{\text{inst}} \setminus F^k_{\text{std}} = \{5821, 16364\}$, we identify latent dimensions specifically enhanced by Instruction following prompts. These reflect both semantic domain shifts (e.g., audio) and structural sensitivities (e.g., punctuation), however they are not related with the concept lowercase at the first glance.

As shown in Tables 12 and 13, both settings share key features such as SAE_{1828} (API-related code) and SAE_{7056} (health-related terminology), indicating persistent themes regardless of instruction. However, the Instruction following setting activates distinct features such as SAE_{5821} (audio production terminology) and SAE_{16364} (conversational punctuation structure), while the non-instruction setting instead emphasizes SAE_{6919} (API error-handling code) and SAE SAE_{15917}(formatted Q&A patterns). We perform constant steering with SAE feature SAE_{5821} and SAE_{16364}, where other top activated features are useless thus regarded.

E.4 Uppercase

Feature-Level Activation Analysis Tables 14 and 15 report the most salient SAE features in each case. Common across both settings are SAE_{1828} (API-related code), SAE_{717} (structured technical content), and SAE_{7056} (health terminology), suggesting stable relevance of these latent features regardless of instruction. Steering with these features would not direct model to generate uppercase responses. The Instruction following condition introduces distinct high-activation features: SAE_{14689} (rules and conditions in user agreements) and SAE_{12687} (legal terminology), which are absent in the non-instruction setting. These may reflect more formal or rigid textual structures often associated with uppercase use cases.

E.5 Language Cases

Feature-Level Activation Analysis across Languages To examine the effect of Instruction following across multilingual prompts, we collect the top-5 SAE features activated for each of the 10 target languages, with and without instruction. As shown in Table 16, several core features—such as SAE_{1828} (API references), SAE_{717} (structured text), and SAE_{7056} (scientific concepts)—appear across nearly all languages and conditions, suggesting shared backbone representations, which are regarded as redundant features and are not used for steering.

Specifically we compute the set difference

Rank	Index	Feature Description	Activation
1	1828	references to API requests and their parameters in code snippets	34.05
2	717	structured elements within technical documentation	26.05
3	6919	code snippets related to API data retrieval and error handling	16.73
4	15917	formatted questions and responses	15.21
5	7056	scientific concepts and terminologies related to health and wellness	13.31

Table 12: Top 5 activated SAE features for Gemma-2-9B-it without instruction (Lowercase)

Rank	Index	Feature Description	Activation
1	1828	references to API requests and their parameters in code snippets	29.14
2	5821	terms and concepts related to audio production and technology	22.65
3	16364	symbols and punctuation marks that indicate structure in conversation, such as question marks, quotes, and parentheses	21.64
4	7056	scientific concepts and terminologies related to health and wellness	15.71
5	717	structured elements within technical documentation	13.61

Table 13: Top 5 activated SAE features for Gemma-2-9B-it with instruction (Lowercase)

Rank	Index	Feature Description	Activation
1	1828	references to API requests and their parameters in code snippets	32.87
2	717	structured elements within technical documentation	28.01
3	6919	code snippets related to API data retrieval and error handling	16.32
4	15917	formatted questions and responses	14.31
5	7056	scientific concepts and terminologies related to health and wellness	14.00

Table 14: Top 5 activated SAE features for Gemma-2-9B-it without instruction (Uppercase)

Rank	Index	Feature Description	Activation
1	14689	phrases indicating rules, conditions, or restrictions regarding user agreements and obligations	27.21
2	1828	references to API requests and their parameters in code snippets	26.11
3	717	structured elements within technical documentation	15.58
4	7056	scientific concepts and terminologies related to health and wellness	15.19
5	12687	specific legal terms and references within a judicial context	11.81

Table 15: Top 5 activated SAE features for Gemma-2-9B-it with instruction (Uppercase)

$F^k_{\text{inst}}(L) \setminus F^k_{\text{std}}(L)$ for each language L and present the results in Table 17. These uniquely activated SAE features reflect dimensions enhanced only under instruction-followed settings. Some, such as SAE_{5679}, appear frequently across languages and may encode general-purpose Instruction following behavior. Others, like SAE SAE_{12251} (German) or SAE_{14869} (Thai), may capture more language-specific structural or formatting behaviors.

E.6 Case V: Instruction with Response Length (Sentence Level)

In this case, we still include the table, omitting the textual explanations of each useful SAE feature in the length constraint case and instead report the corresponding SAE feature indices using cumulative methods. As mentioned before, we include the instructions in the input when performing steering.

Feature-Level Activation Analysis with Response Length Control We examine how re-

Language	SAE Rank = 1	SAE Rank = 2	SAE Rank = 3	SAE Rank = 4	SAE Rank = 5
Arabic (w/o inst.)	1828	6919	717	8688	5549
Arabic (w/ inst.)	6356	5679	717	1828	7056
Bengali (w/o inst.)	1828	6919	717	7197	7056
Bengali (w/ inst.)	6874	1828	5679	31	6098
German (w/o inst.)	1828	717	7056	6919	13295
German (w/ inst.)	12251	1828	717	5679	15917
Persian (w/o inst.)	1828	717	13650	1479	10564
Persian (w/ inst.)	16267	1828	13341	717	5679
Hindi (w/o inst.)	1828	6919	717	15917	208
Hindi (w/ inst.)	1828	7056	5679	31	6098
Italian (w/o inst.)	1828	7056	8985	5549	6919
Italian (w/ inst.)	1828	4427	5679	7056	14174
Swahili (w/o inst.)	1828	717	13295	658	86
Swahili (w/ inst.)	5679	1828	717	2247	12662
Russian (w/o inst.)	1828	15917	6919	7056	12305
Russian (w/ inst.)	5679	15526	1828	11684	7056
Thai (w/o inst.)	1828	717	13295	6919	16152
Thai (w/ inst.)	5679	14869	12335	7056	2764
Vietnamese (w/o inst.)	1828	7056	6919	5549	15917
Vietnamese (w/ inst.)	1828	5679	3579	11684	717

Table 16: Top-5 activated SAE features per language, with and without instruction

Language	Unique SAE Features from Instruction ($F^k_{\text{inst}} \setminus F^k_{\text{std}}$)
Arabic	6356, 5679
Bengali	6874, 5679, 31, 6098
German	12251, 5679, 15917
Persian	16267, 13341, 5679
Hindi	7056, 5679, 31, 6098
Italian	4427, 5679, 14174
Swahili	5679, 2247, 12662
Russian	5679, 15526, 11684
Thai	5679, 14869, 12335, 2764
Vietnamese	5679, 3579, 11684

Table 17: SAE features uniquely activated by instruction across languages ($F^k_{\text{inst}} \setminus F^k_{\text{std}}$)

sponse length control at the sentence level affects internal SAE activations, comparing top-10 features with and without instruction across target lengths (1–3 sentences). As shown in Tables 18 and 19, the non-instruction setting yields identical feature activations across all lengths, indicating that the base model does not inherently modulate internal activations in response to subtle differences in expected output length.

In contrast, Instruction following introduces new activated SAE features—most notably SAE_{9214} and SSAE_{2967} across all lengths. These features are not present in the non-instruction setting and are consistently ranked within the top 5, suggesting they play a central role in sentence-level response control. SAE_{7056} , which is shared across both

Length	SAE Rank = 1	SAE Rank = 2	SAE Rank = 3	SAE Rank = 4	SAE Rank = 5
1 (w/o inst.)	1828	717	6919	15917	7056
1 (w/ inst.)	1828	7056	9214	2967	14983
2 (w/o inst.)	1828	717	6919	15917	7056
2 (w/ inst.)	1828	7056	9214	2967	15917
3 (w/o inst.)	1828	717	6919	15917	7056
3 (w/ inst.)	1828	7056	9214	15917	2967

Table 18: Top 1-5 activated SAE features for length constraint

Length	SAE Rank = 6	SAE Rank = 7	SAE Rank = 8	SAE Rank = 9	SAE Rank = 10
1 (w/o inst.)	5549	208	12122	9214	13295
1 (w/ inst.)	15917	5679	2992	3862	1377
2 (w/o inst.)	5549	208	12122	9214	13295
2 (w/ inst.)	1377	717	6919	4371	2992
3 (w/o inst.)	5549	208	12122	9214	13295
3 (w/ inst.)	6919	1377	717	4371	2992

Table 19: Top 6-10 activated SAE features for length constraint

settings, appears to contribute general linguistic structure, while features like SAE_{9214} and SAE_{2967} likely encode finer-grained length or formatting control.

F SAE Feature Similarity

Since the dimension of SAE steering vectors and MeanActDiff steering vectors are the same, we here define the similarity metrics, with either 1) the internal-similarity between two SAE vectors SAE_i and SAE_j :

$$\text{sim_I}(\text{SAE}_i, \text{SAE}_j) = \frac{\langle \text{SAE}_i, \text{SAE}_j \rangle}{\|\text{SAE}_i\|_2 \|\text{SAE}_i\|_2} \quad (7)$$

or 2) the external-similarity between a pair of SAE_i and MeanActDiff_i for each reasoning case:

$$\text{sim_E} = \frac{\langle \text{SAE}_i, \text{MeanActDiff}_i \rangle}{\|\text{SAE}_i\|_2 \|\text{MeanActDiff}_i\|_2} \quad (8)$$

For clarity, we present two similarity matrices showing the top-5 most activated SAE features for the Gemma-2-2B and Gemma-2-9B models in the math reasoning cases respectively. Furthurmore, we provide a table that reports the similarity metric sim_E for Gemma-2-2B and Gemma-2-9B models. This metric quantifies the alignment between the vectors that trigger Chain-of-Thought (CoT) reasoning behavior and those derived from MeanActDiff steering, helping to explain the observed similarities in their effects.

Overall the similarity between a pair of SAE features is low. This makes sense since SAE training tried to disentangle feature representation. Feature SAE_{2899} and feature SAE_{6782} are positively related other four features. SAE_{1858} and SAE_{2899} share high similarity as observed, and in the Neuronpedia we found that they share the common top activated word "The". The similarity between SAE steering vectors and MeanActDiff steering vectors is somehow correlated, which is shown in the Table 22.

F.1 Instruction Following

We visualize the pairwise cosine similarity of selected SAE steering vectors (Figure 19) and MeanActDiff steering vectors (Figure 20), re-using the similarity metrics from section 5.2.3.

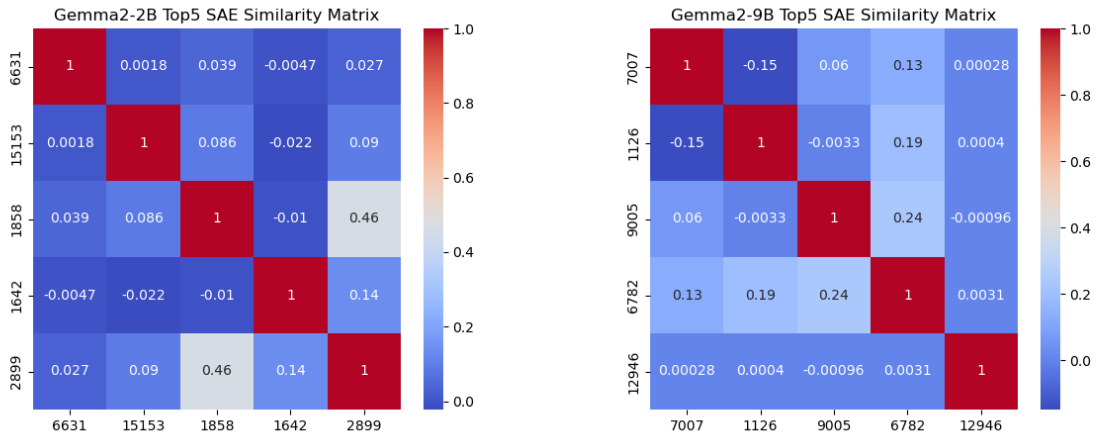
In Figure 19, we observe that the cosine similarity between most SAE features is small and close to zero, suggesting that these features are relatively independent. However, a few pairs such as SAE_{14869} and SAE_{3579}, exhibit moderate similarity (approximately 0.47), indicating some degree of entanglement or functional overlap. SAE_{14869} corresponds

Model Mode	With Instruction	Sentences (Length = 1)	Sentences (Length = 2)	Sentences (Length = 3)	Sentences (Length = 4)
Inference	×	9.30	6.98	9.30	2.33
MeanActDiff	✓	74.42	67.44	60.47	51.16
SAE_{7056}	✓	69.77	58.14	58.14	55.81
SAE_{9214}	✓	69.77	62.79	58.14	58.14
SAE_{2967}	✓	72.09	62.79	60.47	53.49
SAE_{1377}	✓	74.42	62.79	65.12	51.16
Inference	✓	72.09	60.47	58.14	51.16

Table 20: Gemma-2-9B-it: Performance on Instruction Following Dataset IFEval (From At Least to Exactly)

Model Mode	With Instruction	Sentences (Length = 1)	Sentences (Length = 2)	Sentences (Length = 3)	Sentences (Length = 4)
Inference	×	0	4.55	4.55	2.27
MeanActDiff	✓	77.28	70.46	72.73	65.91
SAE_{7056}	✓	65.91	65.91	70.46	63.64
SAE_{9214}	✓	72.73	68.19	63.64	65.91
SAE_{2967}	✓	77.28	72.73	72.73	75.00
SAE_{1377}	✓	70.46	63.64	63.64	70.45
Inference	✓	72.73	72.73	65.91	65.91

Table 21: Gemma-2-9B-it: Performance on Instruction Following Dataset IFEval (From Less Than to Exactly)



(a) Gemma-2-2B Top-5 SAE Feature Similariry Matrix (b) Gemma-2-9B Top-5 SAE Feature Similariry Matrix

Figure 18: Cosine similarity matrices of Top-5 activated SAE features from Gemma-2-2B and Gemma-2-9B models.

Model	Sim_E (max = 1)
Gemma-2-2B	0.2227
Gemma-2-9B	0.3465

Table 22: Cosine similarity between the SAE steering vector that represents CoT reasoning and corresponding MeanActAdd steering vector

to Thai in the SAE latent space, while SAE_{3579} represents Vietnamese. The observed similarity may reflect shared characteristics such as geographical proximity, tonal language structure, and linguistic patterns common to Southeast Asian languages, including subject–verb–object syntax and the use of classifiers.

The MeanActDiff vectors show a much clearer semantic structure. Specifically, steering vectors corresponding to `bn` (Bengali) and `hi` (Hindi) exhibit high cosine similarity (>0.6), which aligns with the linguistic proximity of these languages. This phenomenon suggests that the SAE has learned to group language-related activations in a semantically meaningful way.

We also observe that the vectors for `capital` and `lowercase` have relatively high similarity (>0.5), indicating that these features may encode closely related formatting or syntactic styles. This structured similarity does not emerge as clearly in the raw SAE feature space, further confirming that **MeanActDiff steering surfaces latent semantic**

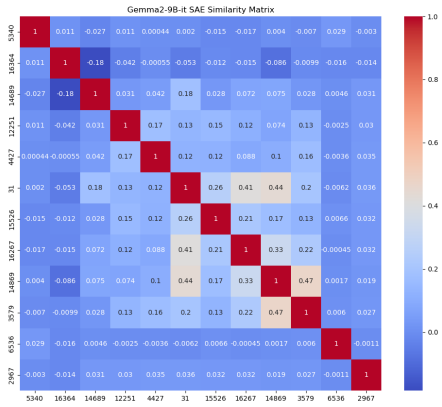


Figure 19: SAE feature similarity matrix

relationships more explicitly, which is the potential reason why MeanActDiff steering is better than SAE steering in lowercase generation.

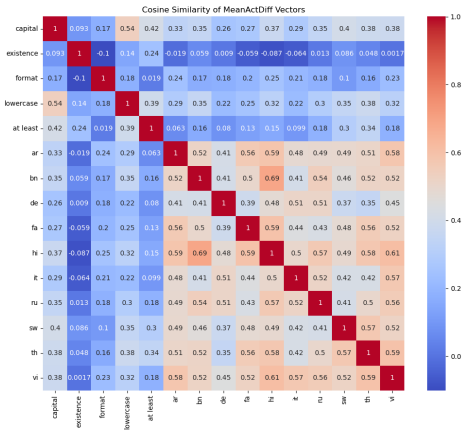


Figure 20: MeanActDiff similarity matrix

Table 23 reports the cosine similarity (`Sim_E`) between the SAE steering vector and its corresponding MeanActAdd steering vector across various instruction cases in the IFEval dataset. A higher similarity indicates that the two steering strategies are capturing consistent directional influence in activation space.

We observe that **format-related tasks**, such as *Uppercase*, *Lowercase*, and *JSON*, all exhibit relatively high similarity scores (ranging from 0.44 to 0.50), suggesting that these syntactic modifications activate similar directions in the model, regardless of the steering method. In contrast, *Length Constraint* matching shows the lowest similarity (0.0985), possibly because it involves stricter token-level constraints that are harder to align in latent space, or because the SAE may not localize the exact-matching behavior as strongly.

Case	Sim_E (max = 1)
Uppercase	0.4938
Lowercase	0.5031
JSON	0.4460
Length Constraint	0.0985
Language-German	0.4649
Language-Italian	0.5209
Language-Hindi	0.3030
Language-Russian	0.3618
Language-Persian	0.4531
Language-Arabic	0.4794
Language-Thai	0.3470
Language-Vietnam	0.3855

Table 23: Cosine similarity between the SAE steering vector and corresponding MeanActAdd steering vector for each case in instruction-followed IFEval dataset

Among language-related instructions, most language pairs exhibit relatively high cosine similarity (typically above 0.45), indicating that the corresponding MeanActDiff vectors activate overlapping regions in the model's representation space. For example, vectors corresponding to *Italian*, *Arabic*, and *Persian* show strong alignment, but even languages such as *Hindi*, *Thai*, and *Vietnamese* maintain moderately high similarity with others. This suggests a shared latent structure across language steering signals. The slight variability may be attributed to differences in tokenization coverage, script diversity, or activation sparsity associated with each language.

G Prompt Template

This section outlines the prompt template used in instruction-following tasks. These templates enforce strict formatting and response structures to encourage model consistency.

For math reasoning prompts, we include 8-shot chain-of-thought examples inside a shaded gray box preceding the query to improve multi-step reasoning:

Table 24: Instruction examples for different tasks.

Instruction	Example
Lowercase	Please ensure that your response is in English, and in all lowercase letters.
Language	Please respond using only the German language, no other language is allowed.
JSON Format	Wrap the entire output in JSON format.
English Capitalize	Make sure your entire response is in English, and in all capital letters.
Sentence Frequency	Answer with exactly 3 sentences.
Word Inclusion	Please include the word cat in the response.

Few-shot Chain-of-Thought Examples

Question: Olivia has $23. She bought five bagels for $3 each. How much money does she have left? Answer: Olivia had 23 dollars.
She bought 5 bagels at 3 dollars each: 5 × 3 = 15.
She spent 15 dollars, so she has 23 - 15 = 8 dollars left.
The answer is 8.

Question: There were nine computers in the server room. Five more computers were installed each day, from monday to thursday. How many computers are now in the server room?
Answer: There were originally 9 computers.
5 computers were added each day for 4 days: 5 × 4 = 20.
So, total is 9 + 20 = 29.
The answer is 29.

Question: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
Answer: There were originally 15 trees.
After planting, there were 21 trees.
So, 21 - 15 = 6 trees were planted.
The answer is 6.

Question: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
Answer: There were 3 cars.
2 more cars arrived: 3 + 2 = 5.
The answer is 5.

Question: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?
Answer: Jason had 20 lollipops.
After giving some away, he had 12.
So he gave away 20 - 12 = 8.
The answer is 8.

Question: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?
Answer: Shawn had 5 toys.
He got 2 from mom and 2 from dad: 2 + 2 = 4.
So he has 5 + 4 = 9 toys.
The answer is 9.

Question: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
Answer: Leah had 32 and her sister had 42: 32 + 42 = 74.
They ate 35: 74 - 35 = 39.
The answer is 39.

Question: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?
Answer: Michael started with 58.
Lost 23 on Tuesday: 58 - 23 = 35.
Lost 2 more on Wednesday: 35 - 2 = 33.
The answer is 33.

H Simple Evaluation/Verification with Visualization

Neuronpedia (Lin, 2023) is an interactive web-based platform that hosts labeled SAE feature dictionaries for open-weight models. Each SAE feature is tagged with human-interpretable descriptions (e.g., "list start", "closing brace", "polite language") based on automated annotation pipelines by gpt 4o. We utilize Neuronpedia to investigate and verify the semantics of key SAE latents identified during steering experiments. Specifically, we validate whether activation-based interventions correspond to meaningful internal concepts.

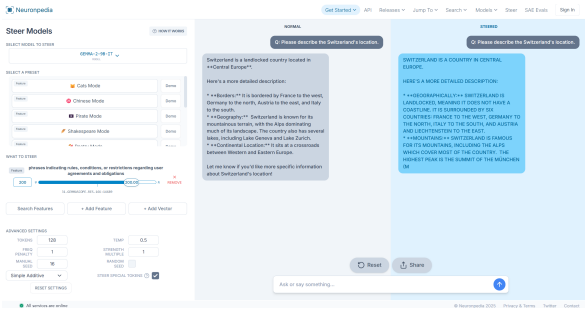


Figure 21: Example 1 by Neuronpedia: Steering models to generate uppercase answers by SAE steering vectors

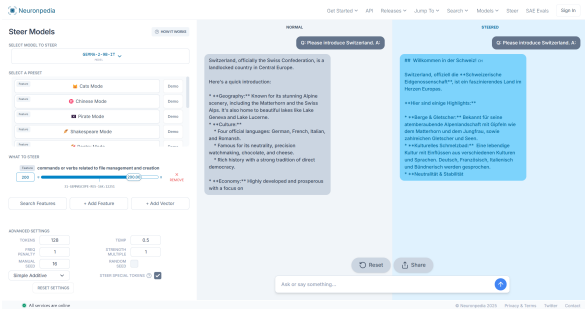


Figure 22: Example 2 by Neuronpedia: Steering models to generate German answers by SAE steering vectors

I Linear Probing with SAE Feature Gating

I.1 Methods

To quantitatively assess whether specific SAE features encode concepts related to arithmetic, we employ a linear probing methodology. The objective is to train a simple linear classifier to decode a specific arithmetic operation (addition, subtraction, multiplication, or division) from the internal representations of a language model. The performance of this probe serves as a measure of how linearly separable the target information is within the model's activations.

Our probing setup introduces a key modification to the standard approach by using an SAE feature vector as a gating mechanism. For a given input sequence representing an arithmetic expression, we first extract the sequence of hidden state activations, $\mathbf{H}$, from a target layer of the frozen language model. Instead of probing these activations directly, we modulate them via an element-wise Hadamard product ($\odot$) with a chosen SAE feature vector, $\mathbf{d}_{l,j}$:

$$\mathbf{H}_{\text{gated}} = \mathbf{H} \odot \mathbf{d}_{l,j} \tag{9}$$

This operation selectively amplifies or suppresses dimensions in the activation space that are aligned with the feature captured by the SAE vector.

The resulting sequence of gated activation vectors, $\mathbf{H}_{\text{gated}}$, is then aggregated into a single vector representation through mean-pooling across the token dimension. This final vector is fed into a linear classifier, which is trained to output logits for the four operation classes. A softmax function is applied to these logits to produce a probability distribution. The classification accuracy of this probe indicates the extent to which the specific SAE feature $\mathbf{d}_{l,j}$ encodes information relevant to the arithmetic operation being classified.

I.2 Results

Table 25: Probing with SAE 15153 and Gemma-2-2B

	precision	recall	f1-score
+	0.33	0.41	0.36
-	0.23	0.29	0.26
*	0.55	0.21	0.31
/	**0.72**	**0.82**	**0.76**
macro avg	0.46	0.43	0.42

Table 26: Probing with SAE 1642 and Gemma-2-2B

	precision	recall	f1-score
+	**0.44**	**0.56**	**0.49**
-	0.29	0.19	0.23
*	0.38	0.36	0.37
/	0.42	0.47	0.44
macro avg	0.38	0.40	0.38

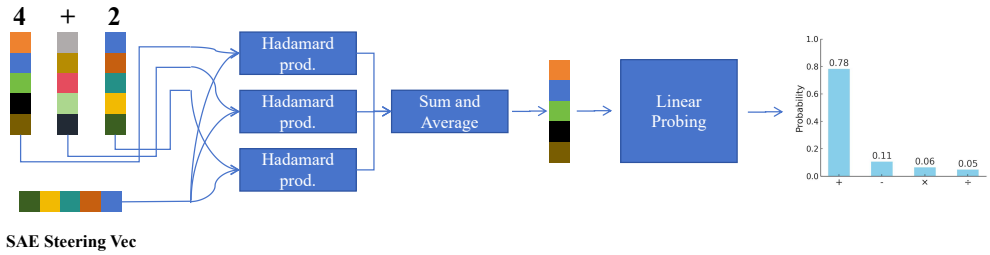


Figure 23: Linear Probing

Table 27: Probing with SAE 435 and Gemma-2-2B

	precision	recall	f1-score
+	0.77	**0.87**	**0.82**
-	0.55	0.43	0.48
*	0.50	0.52	0.51
/	**0.78**	0.83	0.81
macro avg	**0.65**	**0.66**	**0.65**

Table 28: Probing with MeanActDiff Steering Vector and Gemma-2-2B

	precision	recall	f1-score
+	0.30	0.11	0.16
-	0.43	0.89	0.58
*	0.72	0.14	0.23
/	**0.67**	**0.94**	**0.78**
macro avg	0.53	0.52	0.44

Table 29: Probing with SAE 7007 and Gemma-2-9B

	precision	recall	f1-score
+	0.41	0.76	0.53
-	0.11	0.05	0.07
*	0.78	0.33	0.47
/	**0.71**	**0.94**	**0.81**
macro avg	**0.50**	**0.52**	**0.47**

Table 30: Probing with SAE 9005 and Gemma-2-9B

	precision	recall	f1-score
+	0.34	0.75	0.47
-	0.42	0.15	0.22
*	0.46	0.04	0.07
/	**0.67**	**0.91**	**0.77**
macro avg	0.47	0.46	0.38

Table 31: Probing with SAE 6782 and Gemma-2-9B

	precision	recall	f1-score
+	0.45	0.75	0.56
-	0.46	0.25	0.33
*	0.36	0.17	0.23
/	**0.67**	**0.89**	**0.77**
macro avg	0.49	**0.52**	**0.47**

Table 32: Probing with MeanActDiff Steering Vector and Gemma-2-9B

	precision	recall	f1-score
+	0.45	0.75	0.56
-	0.24	0.15	0.18
*	0.72	0.22	0.34
/	**0.57**	**0.80**	**0.67**
macro avg	0.49	0.48	0.44

As shown in Tables 27 through 32, the linear

probing results reveal notable differences in performance across various SAE steering vectors. These experiments are conducted on synthetic data specifically designed around basic arithmetic operations, including addition, subtraction, multiplication, and division. As a result, SAE vectors that encode operation-specific information such as SAE_{435} and SAE_{7007}) tend to achieve higher probing accuracy, with macro-average F1-scores of 0.65 and 0.47, respectively.

In contrast, SAE vectors like SAE_{15153}, previously shown to be effective in generating complex Chain-of-Thought reasoning or instruction-following behaviors, achieve lower probing scores with macro F1-score of 0.42. This suggests that their activation patterns are better aligned with higher-level reasoning rather than surface-level operation classification. This suggests that SAE features specialized for basic symbolic manipulation are more effective for linear probing on arithmetic-focused datasets, whereas reasoning-related SAE vectors are less suited for such tasks due to their reasoning nature.