MEMORY RETRIEVAL AND CONSOLIDATION IN LARGE LANGUAGE MODELS THROUGH FUNCTION TOKENS

Anonymous ACL submission

Abstract

The remarkable success of large language models (LLMs) stems from their ability to consolidate vast amounts of knowledge into the memory during pre-training and to retrieve it from the memory during inference, enabling advanced capabilities such as knowledge memorization, instruction-following and reasoning. However, the mechanisms of memory retrieval and consolidation in LLMs remain poorly understood. In this paper, we propose the function token hypothesis to explain the workings of LLMs: During inference, function tokens activate the most predictive features from context and direct next token prediction (memory retrieval). During pre-training, predicting the next tokens (usually content tokens) that follow function tokens increases the number of learned features of LLMs and updates the model parameters (memory consolidation). Function tokens here roughly correspond to function words in linguistics, including punctuation marks, articles, prepositions, and conjunctions, in contrast to content tokens. We provide experimental evidence supporting this hypothesis. Using bipartite graph analysis, we show that a small number of function tokens activate the majority of features. Case studies further reveal how function tokens activate the most predictive features from context to direct next token prediction. We also find that during pre-training, the training loss is dominated by predicting the next content tokens following function tokens, which forces the function tokens to select the most predictive features from context.

1 Introduction

Large Language Models (LLMs) (OpenAI, 2022; Brown et al., 2020; OpenAI, 2023; Anthropic, 2024; Team et al., 2023; Liu et al., 2024) have demonstrated remarkable capabilities. They possess strong knowledge memorization abilities, ranging from remembering simple factual knowledge (e.g., *The capital of the United States is Washington, D.C.*) to the verbatim reproduction of lengthy passages (e.g., *Recite Martin Luther King Jr's "I Have a Dream" speech word by word*). Beyond that, LLMs also exhibit strong general skills, such as instruction following (Ouyang et al., 2022; Wei et al., 2021) (e.g., *As a financial analyst: explain quantitative tightening, then list three stock market impacts.*) and reasoning (Wei et al., 2022; Kojima et al., 2022) (e.g., *The streets are wet and the sidewalks are slick. What is the most likely explanation?*).

In the human brain, long-term memory forms through synaptic consolidation, where the synapses between neurons are strengthened, ultimately creating neural circuits that store knowledge (Josselyn and Tonegawa, 2020). Inspired by this biological mechanism, artificial neural networks have been developed. These systems consist of neurons linked by weighted connections, and their weights (parameters) are obtained by training on data. The weights of a neuron determines how it responds to its inputs to produce an activation (Geva et al., 2021). A technique utilizing Sparse Autoencoders (SAEs) (Cunningham et al., 2023) has been developed recently to analyze Transformer-based LLMs (Vaswani et al., 2017). It enables the decomposition of neuron activations into interpretable features, providing insights into how the circuits within the Transformer's layers are composed of these interpretable features (Elhage et al., 2022; Chen et al., 2025b; Hendel et al., 2023). Despite significant progress in understanding LLM neuron activations, the memory mechanisms remain poorly understood. In particular, two fundamental questions are still not well addressed: (1) How is the memory retrieved during inference? and (2) How is the memory consolidated during pre-training? In this paper, we present our investigation into these questions. We find that analyzing from the perspective of function tokens and content tokens can help unravel the mystery of memory retrieval and

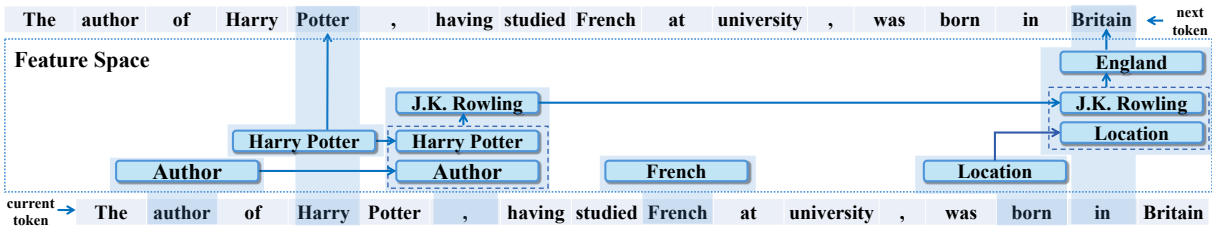


Figure 1: Function tokens can dynamically activate the most predictive features from the context to guide the next-token prediction. For example, the function token 'in' reactivates features 'J.K. Rowling' and 'Location' from context (while suppressing feature 'French') and activates 'England' to predict 'Britain'. In contrast, the content token 'Harry' activates feature 'Harry Potter'.

memory consolidation.

In linguistics, function words are words that have little semantic meanings but play crucial grammatical and connective roles within and between sentences, such as articles, prepositions, and conjunctions (Carnap, 2014). In contrast, content words are words that convey semantically explicit and rich meanings. The distribution of words in natural language follows Zipf's law (Kanwal et al., 2017). In this distribution, function words occur with disproportionately high frequencies, occupying the head, while content words appear with much lower frequencies, forming the long tail. LLMs utilize tokens, which may represent words, sub-words, or punctuation marks. In our work, for ease of experimentation, we automatically classify tokens into 'function tokens' and 'content tokens' based on their frequencies in the pre-training corpus, using this as an approximation of the linguistic concepts.

To investigate the role of function tokens during inference, we construct a bipartite graph connecting tokens to features obtained via SAE decomposition. We show that, although few in number, function tokens activate a large proportion of the LLM's features. Furthermore, our case studies show that the activation patterns for function tokens differ from those for content tokens. Function tokens dynamically reactivate predictive features from the context, whereas content tokens show little evidence of this effect. To understand why feature activations are centered on function tokens, we conduct pre-training experiments. We track next-token prediction loss across four categories based on whether the current token and the next token are function or content tokens. We find that LLMs first learn to predict function tokens before gradually learning to predict content tokens, a process accompanied by an increase in the number of features and the learning of the parameters. Furthermore, pre-training is dominated by the prediction of content tokens that follow function tokens. These observations reveal why function tokens can access a large portion of the LLM's features. Based on these findings, we propose the Function Token Hypothesis (see an example in Figure 1).

In this paper, the LLMs are GPT-type models with a Transformer decoder architecture, obtained through pre-training and post-training (including SFT and RL) (Nakano et al., 2021; Radford et al., 2018). Both pre-training and inference are conducted autoregressively via next-token prediction. At each layer of the Transformer, a vector of activations (after the add-norm operation of FFN) can be created. SAE can be performed on this activation vector to obtain a linear combination of features. Here, knowledge refers to the LLM's parameters as well as all possible features that can be derived from them. Memory is the virtual system that stores the knowledge. Memory retrieval means the activations of features and circuits (Olah et al., 2020; Elhage et al., 2021; Wang et al., 2023; Merullo et al., 2024), while memory consolidation means the learning of the parameters to form and expand features and circuits.

Function Token Hypothesis. During inference, *function tokens activate the most predictive features from context and direct next token prediction (memory retrieval). During pre-training, predicting content tokens based on the function tokens drives the LLM to update its parameters to learn and expand features (memory consolidation).*

The function token hypothesis is also supported by many phenomena observed in LLM research. For example, activations with unusually large magnitudes often occur at the initial tokens, periods, or newlines (Sun et al., 2024). Meaningless separator tokens disproportionately affect attention compared to semantically rich tokens (Chen et al., 2025a). The use of 'pivot tokens' during post-training can significantly enhance performance in response (Abdin et al., 2024). Training that concentrates on high-entropy tokens also yields better

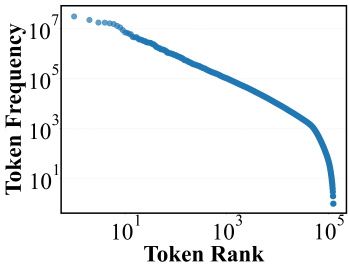
performance (Wang et al., 2025a). We argue that these tokens are all function tokens that behave as the hypothesis predicts.

We believe that unraveling the important role of function tokens in LLM memory mechanisms not only enhances research on LLM interpretability but also provides insights for designing advanced learning algorithms, particularly those for enhancing alignment with human values.

We summarize our main contributions:

- We demonstrate that during inference, function tokens activate the most predictive features from context and direct next token prediction.
- We show that feature growth during pre-training is driven by the prediction of content tokens that follow function tokens.
- We propose the Function Token Hypothesis for explaining LLM memory mechanisms.

2 Preliminary



(a) Zip'f distribution.

Token rank	Token text	Token Fraction	Cumulative Fraction	Document Coverage
1	,	3.60%	3.60%	95.00%
2	the	3.19%	6.79%	90.92%
3	.	2.23%	9.10%	95.80%
4	and	1.81%	10.91%	89.69%
5	of	1.80%	12.71%	87.59%
6	to	1.68%	14.40%	88.71%
7	white space	1.59%	15.99%	81.35%
8	a	1.33%	17.32%	87.62%
9	in	1.16%	18.48%	86.04%
10	.\n	0.90%	19.39%	84.58%

(b) The 10 most frequent tokens.

Figure 2: Token frequency statistics in SlimPajama.

2.1 Model Memory and Superposition

In a neural network, weights constitute the substrate of memory, while activations serve as the medium through which this memory is retrieved in response to a given input. Recent work on LLM interpretability shows the phenomenon of superposition (Elhage et al., 2022), in which features can be extracted from the activations of neurons in a Transformer-based LLM. The number of extracted features usually far exceeds the number of neurons. There exist many polysemantic neurons, each of which represents multiple meanings.

Through sparse dictionary learning, polysemantic neuron activations can be decomposed into monosemantic features, each corresponding to a distinct, human-interpretable concept, such as the Golden Gate Bridge (Templeton et al., 2024). A widely used method for dictionary learning is the Sparse Autoencoder (SAE), which learns to linearly decompose neuron activations through a reconstruction task. SAE decomposes an activation $\mathbf{y} \in R^d$, typically the output of a specific layer, into a linear combination of features:

$$\mathbf{y} = \sum_{i=1}^{n} c_i \mathbf{f}_i = c_1\mathbf{f}_1 + c_2\mathbf{f}_2 + ... + c_n\mathbf{f}_n. \quad (1)$$

Here, c_i is the coefficient for feature $\mathbf{f}_i \in R^d$, the set of features $\{\mathbf{f}_i\}$ constitutes an approximately orthogonal basis, and n represents the number of features, typically $n \gg d$.

Furthermore, the behavior of the LLM during generation can be partially controlled by steering the activations of features. For example, steering can control both specific concepts (e.g., Golden Gate Bridge (Templeton et al., 2024)) and behavioral patterns (e.g., sycophantic behavior (Panickssery et al., 2023)). Similar feature activation phenomena are observed in human memory recall, with empirical evidence supporting the existence of neurons representing specific or general concepts.

2.2 Function Tokens and Content Tokens

We tokenized the SlimPajama-627B corpus (Soboleva et al., 2023), a widely used pre-training dataset, using the LLaMA-3.1 tokenizer and sampled 1 billion tokens for statistical analysis. We group the tokens into function tokens and content tokens based on their frequency. This leverages the linguistic fact that function words typically have higher frequency, while content words have lower frequency. Starting from the most frequent, we add tokens until the set covered 40% of all token occurrences, yielding 122 tokens labeled as function tokens; the rest are taken as content tokens. The resulting set of function tokens roughly corresponds to the function words defined in linguistics, with several exceptions like punctuation marks. The full list of function tokens appears in Appendix C.

Token Frequency and Zipf's Law As shown in Figure 2a, token frequency follows the Zipf's law (Piantadosi, 2014): $f(r) \propto r^{-\alpha}$, where $f(r)$ is the frequency of the token ranked r, revealing a

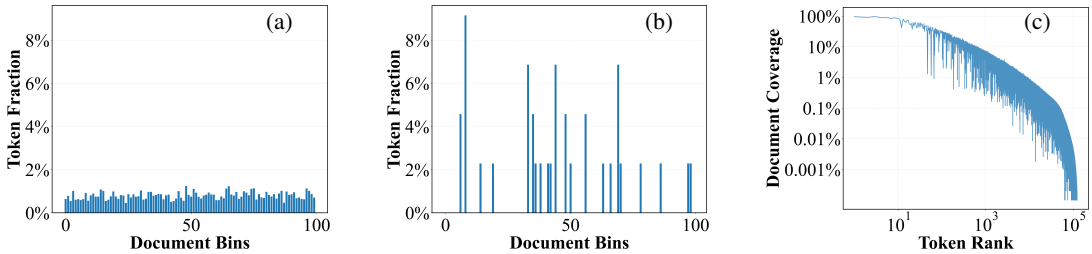


Figure 3: Distribution of function and content tokens. Document bins represent equal partitions of corpus. (a) Function token 'of' shows uniform, dense coverage. (b) Content token 'Tokyo' shows sparse coverage. (c) Document coverage vs. token rank follows a power-law.

Token		Cumulative Feature Coverage		
Rank	Text	Layer 9	Layer 20	Layer 31
1	.	23.19%	51.32%	37.21%
2	,	32.01%	62.45%	49.78%
3	the	36.88%	66.93%	55.15%
4	\n	39.68%	71.30%	59.86%
5	and	41.21%	71.97 %	61.48%
6	to	43.16%	73.07 %	63.30%
7	of	46.00%	74.43 %	65.16%
8	white space	47.44%	75.70 %	67.08%
9	a	47.96%	76.12%	67.74%
10	in	48.52%	76.46%	68.27%

Table 1: Cumulative feature coverage by top-10 tokens.

fundamental property of natural language: a few tokens are used frequently, while most are used infrequently. For example, the 10 most frequent tokens account for 19.39% of the corpus (Figure 2b).

Document Coverage A pre-training corpus contains a vast number of documents. High-frequency tokens are distributed uniformly across documents, while low-frequency tokens appear frequently within a limited number of documents (Rychlỳ, 2011), showing bursty distributions. For instance, as shown in Figure 3, the function token 'of' appears with similar frequency across documents, whereas the content token 'Tokyo' occurs only in a few. Crucially, uniform document coverage means function tokens are exposed to a wide range of semantic contexts during pre-training and inference, whereas the bursty distribution of content tokens confines them to a narrow set of contexts. This contextual asymmetry, rather than raw frequency itself, is what we will show drives the asymmetric activation patterns in §3.1.

As shown in Figure 3, token frequency strongly correlates with document coverage.Figure 2b lists the 10 most frequent tokens along with their document coverage. Formally, the document coverage of token t is $\frac{|\{d \in D : t \in d\}|}{|D|}$, where D denotes the entire set of documents. The 40% threshold for selecting function tokens is chosen to balance coverage and purity rather than arbitrarily. Increasing this threshold (e.g., to 50%) introduces approximately 300 additional token types, many with extremely low document coverage ($< 10\%$), reflecting the bursty behavior shown in Figure 3.

3 Memory Retrieval through Function Tokens

Building on the contextual asymmetry identified in §2.2, we examine whether function tokens correspondingly access a more diverse region of the model's feature space. We construct a token-feature bipartite graph and find that, although a content token may occur many times in the corpus, it activates only a narrow set of features tied to its bursty contexts; in contrast, a small number of function tokens, by virtue of being exposed to diverse contexts, activate the majority of features in the model. In other words, function tokens are highly polysemantic, while content tokens are largely monosemantic.

3.1 A Few Function Tokens Activate Most Features

We use Gemma2-9B (Team et al., 2024) with its open-source SAEs (Gemma Scope) (Lieberum et al., 2024), selecting the largest available dictionary width 2^{20} for comprehensive feature decomposition. Within a single prompt, each token activates 200 features (the L0), with similar per-occurrence activation counts for function and content tokens.

To obtain a global picture of token-feature relationships, we aggregate activations over a large number of diverse prompts and construct a token-feature bipartite graph. As illustrated in Figure 5, construction involves the following steps: (1) Extract activations. We feed 10,000 randomly sampled raw documents from the SlimPajama validation dataset into Gemma2-9B, with approximately 5 million tokens, and extract activations from the residual stream. We focus on three representative layers: layer 9 (shallow), layer 20 (middle), and

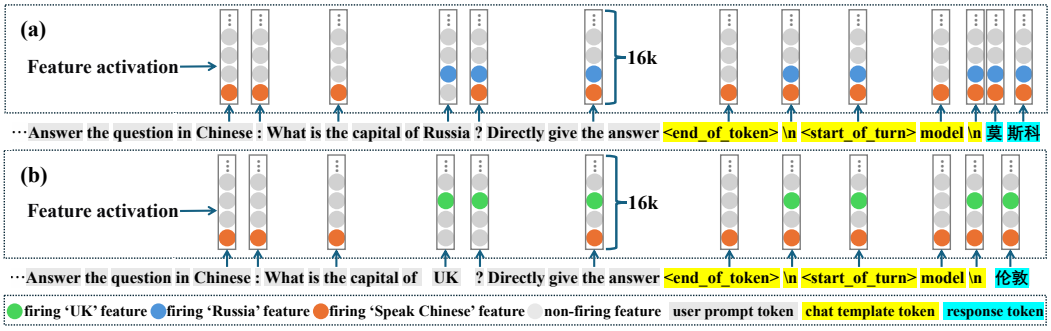


Figure 4: Function tokens can dynamically reactivate predictive features based on different contexts.

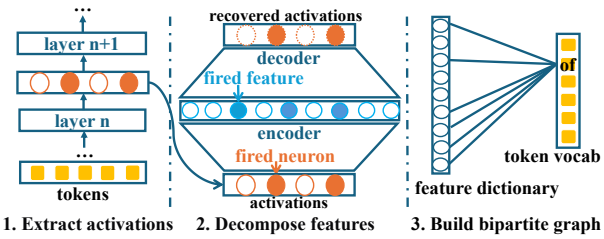


Figure 5: Construction of the bipartite graph.

layer 31 (deep). (2) Decompose features. For each layer, we apply the corresponding SAE to decompose token activations into sparse features. (3) Build bipartite graph. A token is linked to a feature if it activates the feature in a context. Each token-feature pair is connected by at most one edge, regardless of how many times the activation occurs.

The bipartite graph comprises token nodes (equal to the vocabulary size) and feature nodes (2^{20}, the SAE dictionary width). An edge between a token and a feature indicates that the token activates the feature in at least one context. Over 87% of dictionary features are activated in each of the three layers, indicating broad feature coverage. Token degree thus measures how broadly a token can access the feature space across contexts.

Figure 6 presents the degree of each token in the token-feature bipartite graph. The results reveal that a small set of function tokens can activate most features. Table 1 shows that the top 10 frequent tokens alone account for a substantial proportion of feature activations. In particular, in the middle layer, known to be the most expressive and interpretable (Panickssery et al., 2024; Soligo et al., 2025; Chen et al., 2025b), these tokens can activate more than 70% of the features, demonstrating function tokens' universal access to the feature space. This is a striking inversion of the linguistic intuition: function tokens traditionally regarded as semantically light turn out to be the broadest gateways into the model's representational space.

3.2 Feature Reactivation via Function Tokens

Why can a small number of function tokens activate most features? We hypothesize that function tokens can reactivate the most predictive features, based on preceding contexts.

We design an experiment to examine this hypothesis. First, we identify three interpretable features in Gemma2-9B-it (Team, 2024): Feature 15261 corresponds to 'Speak Chinese', Feature 9591 corresponds to 'Russia', and Feature 13751 corresponds to 'UK'. The approach for identifying interpretable features is described in Appendix A. We then examine their activations during inference. We employ the following prompt template, wrapping the chat template used in Gemma2-9B-it.

We evaluate two prompts and record each token's feature activations. **Prompt 1**: *Answer the question in Chinese: What is the capital of Russia?* **Prompt 2**: *Answer the question in Chinese: What is the capital of UK?* As shown in Figure 4, for Prompt 1, the 'Speak Chinese' feature is first activated by the token 'Chinese', and the 'Russia' feature by the token 'Russia'. Function tokens such as ':', 'the' and '\n' act as conduits for propagating and re-creating these activations. Prompt 2 shows the same pattern. The only change between the prompts is replacing 'Russia' with 'UK', while the same function tokens orchestrate different feature combinations, resulting in distinct model outputs.

4 Memory Consolidation through Function Tokens

We analyze how memory consolidation occurs during pre-training. We train two models and track their losses on function and content tokens, as well as their feature growth. We find that (1) learned features increases with training steps; (2) subsequently, the optimization process becomes dominated by learning to predict content tokens, especially pre-

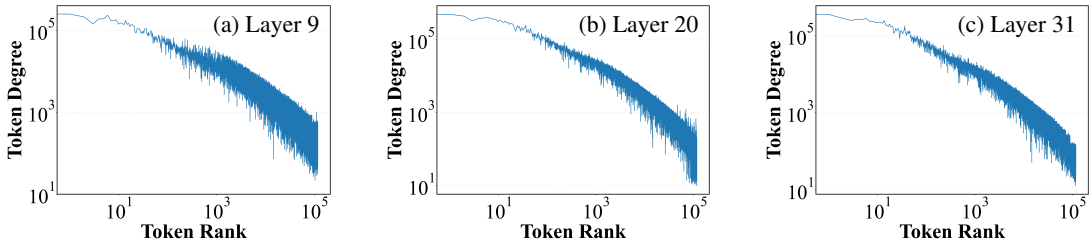


Figure 6: Token degrees in the bipartite graph, token ranked by frequency from the sampled data.

Prompt Template

<bos><start_of_turn>user
{prompt} Directly answer the question<end_of_turn>
<start_of_turn>model

dicting content tokens that follow function tokens.

4.1 Pre-Training Setup

We train two models from scratch using the LLaMA-3.1-8B (Grattafiori et al., 2024) architecture: an 8B model with the originial 32 layers and a 1.5B models with only 2 layers, keeping other components unchanged. We use SlimPajama-627B (Soboleva et al., 2023) as our pre-training corpus, which is a diverse, high-quality collection of web data that has been carefully deduplicated and filtered. This dataset is well-suited for studying memory consolidation during pre-training. We train for one complete epoch over its 627 billion tokens. We replicate the training hyperparameters of LLaMA-3.1-8B for reproducibility: batch size 1024, max sequence length 4095, AdamW optimizer. The learning rate warm up linearly for 8,000 steps to 8×10^{-5}, then decays by cosine annealing to 8×10^{-7}. Training runs on 128 GPUs with 80GB memory each.

4.2 Memory Consolidation as Feature Formation

Due to computational constraints, we perform feature decomposition only on the 1.5B model. To track the number of emergent features during pre-training, we train SAEs on second-layer activations at multiple checkpoints. We use JumpReLU-SAE (Rajamanoharan et al., 2024b) with a tanh penalty function (Bloom et al., 2024), wich outperforms alternatives such as TopK-SAE (Gao et al., 2025) and Gated-SAE (Rajamanoharan et al., 2024a). Training details are in Appendix B.

We select three representative pre-training checkpoints for SAE training: 3000 steps, 50,000 steps and 130,000 steps, corresponding to early, intermediate and late stages of pre-training. For each checkpoint, we sample text sequences from SlimPajama to obtain 500,000 activations, which are input to the SAE to count the total number of decomposed unique features. As shown in Figure 7, the number of features grows substantially over the progress of pre-training, reflecting the model's increasing representational capability and corresponding to memory consolidation.

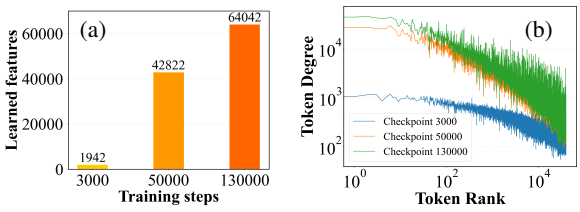


Figure 7: (a) Growth of learned features across training steps. (b) Token degree evolution by checkpoints.

4.3 Loss on Function and Content Tokens

To track loss changes for function and content tokens, we categorize next-token prediction of the form $p(\text{next token} \mid \text{current token}, \text{context})$ into four groups based on whether the current and next tokens are function or content tokens. For example, $p(\text{next token} = \text{function token} \mid \text{current token} = \text{function token}, \text{context})$ is denoted as function→function. The other three categories are defined similarly: function→content, content→function, and content→content. Figure 8 presents the pre-training loss curves of the four groups for both the 1.5B and 8B models, along with the average loss curve across all tokens. We highlight several key observations.

Function→Content is the hardest prediction task. During pre-training, the function→content group exhibits the highest loss in both the 1.5B and 8B models, indicating that predicting content tokens conditioned on a preceding function token is the hardest of the four transitions. This is consistent with the intuition that function tokens carry limited lexical information about the upcoming content, forcing the model to retrieve predictive information from the longer-range context. Figure 9 provides an example to show how LLM generation evolves

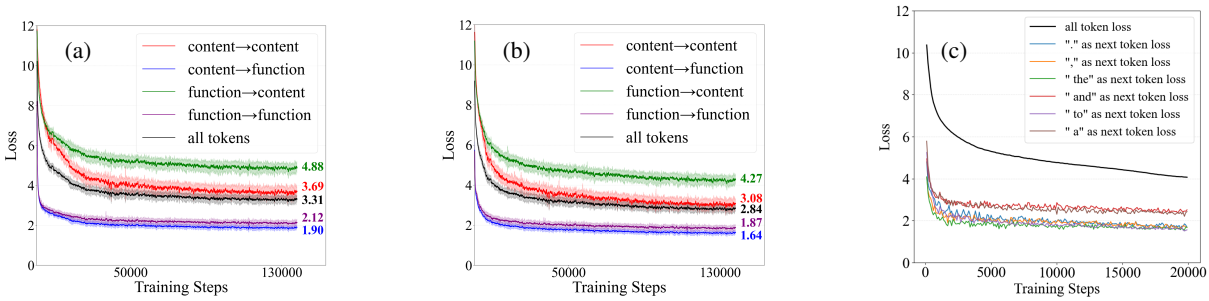


Figure 8: Pre-training loss curves of different token groups.(a) Grouped token loss for 1.5B model. (b) Grouped token loss for 8B model. (c) Typical function token losses in 1.5B model.

during pre-training. At early stage (step 100), generation are random. By step 3,000, the model predicts only function tokens (e.g., 'the', 'a'). By step 50,000, it generates locally coherent phrases like 'learning to' and 'to be'. More complex predictions requiring long-range dependencies, emerge later in the 8B model. For instance, correctly predicting the next token after '...struggle with' requires recalling the earlier context 'learning to read'.

Function tokens cover the majority of the feature dictionary at every checkpoint. Using the bipartite graph analysis described in Section 3.2, we track how token-feature activations evolve across checkpoints. As shown in Figure 7, the total number of features grows monotonically with training. Throughout training, function tokens activate nearly the entire dictionary at every checkpoint, whereas content-token coverage is progressively diluted as the dictionary expands.

Function tokens drive feature formation. Function→Content is the hardest case, since a function token itself carries little lexical information. In the forward pass, the model must select predictive features from preceding context activations to predict the next content token. In the backward pass, these paths carry the long-range next-token gradient signals back to the selected features and drive their updates. The loss stays high without saturating, supplying a persistent gradient for feature updates. Across the checkpoints we examined, the feature dictionary keeps expanding, yet function tokens connect to most features across all checkpoints, where as content tokens each touch only a narrow slice, indicating that features grow through function tokens.

5 Function Token Hypothesis

Our experiments suggest that function tokens are crucial for memory consolidation and memory retrieval in LLMs, leading to our Function Token Hypothesis. During inference, function tokens activate the most predictive features from context and direct next token prediction (memory retrieval). During pre-training, function tokens drive the formation of new features as the feature dictionary expands (memory consolidation).

We postulate that the function token hypothesis is the compound result of four factors in LLM training: the training loss (cross entropy loss), learning algorithm (SGD (Ruder, 2017) or backpropagation (Rumelhart et al., 1986)), model architecture (Transformer), and nature of language data.

LLM training is driven by next token prediction. Minimizing the loss means making predictions as accurate as possible. (Minimizing the total loss over all next tokens in the training data is equivalent to compressing it as compactly as possible (Delétang et al., 2024).) During training, SGD always reduces the training loss the most by computing and utilizing the steepest descent.

Each Transformer (decoder-only) block consists of a self-attention layer followed by an FFN layer. The self-attention layer composes activations from different positions in the context (note that compositionality is the key characteristic of language (Chomsky, 2002)). while the FFN layer stores knowledge in its parameters and produces the activation at each position, and features can be extracted from the activations.

A natural language text is always segmented by function tokens. From each function token to one of its preceding function tokens, a chunk exists, extending until the beginning of the text. These chunks can represent a phrase, a sentence, or a paragraph, and they are nested. When the LLM's prediction reaches the token immediately following a function token, this implies the start of predicting the next chunk; the task is far more challenging, as it requires understanding the meaning of the entire context up to that point. It compels the LLM to activate the most predictive features in the context

Model Size	Training Steps	When	young	children	are	learning	to	read	,	they	often	struggle	with	complicated	words
1.5b	100 steps (0.7%)	sharing	SUCCESS	GHz	liament	").	liament	ブ	Liter	follower	readOnly	readOnly	posites	튜	になり
	3000 steps (2.3%)	,	a	,	a	to	the	the	and	are	,	,	the	and	,
	50000 steps (38%)	the	young	are	in	to	the	the	and	are	find	to	the	ideas	and
	130000 steps (94%)	the	young	are	in	to	read	the	they	are	have	to	the	language	and
8b	130000 steps (94%)	the	people	are	not	to	read	,	they	are	have	to	reading	words	and

Figure 9: Next token predictions at different training steps. The first row shows the prompt. Each subsequent row shows the next-token predictions conditioned on all preceding tokens. For example, the third column uses 'When young' as input, and the fourth uses 'When young children' as input.

during training and reactivate the most predictive features during inference.

6 Related Work

Research on neural memory dates back to the Hopfield network (Hopfield, 1982), also known as associative memory network, which consolidates memories by adjusting weights between neurons. Hopfield networks have evolved into restricted Boltzmann machines (Fischer and Igel, 2012) and feed-forward networks that utilize key-value memories (Geva et al., 2021). Recent research on superposition (Elhage et al., 2022) has shown that it is possible to uncover the features of neural networks such as Transformer, where the number of features is much larger than that of neurons. Through dictionary learning, activations can be decomposed into monosemantic features. Existing work has demonstrated that such decomposed features can effectively steer model behaviors (Chen et al., 2025b; Panickssery et al., 2024), by maintaining specific feature activations, controlling access to memories, and directing the model through the generation.

Existing research on LLMs has identified important patterns involving function tokens. For example, separator tokens produce large activations (Sun et al., 2024) and distinct attention weights (Chen et al., 2025a), enabling efficient KV cache designs retaining only separator caches. The crucial role of "formatting" in post-training is also widely recognized (Zhou et al., 2023; Ye et al., 2025; Mamidanna et al., 2025; Li et al., 2025), a function primarily controlled by tokens such as '\n'. Furthermore, recent work on reinforcement learning for reasoning finds that training primarily on high-entropy tokens like 'thus' improves performance (Wang et al., 2025b), while Phi-4 (Abdin et al., 2024) identifies 'pivot tokens', often following function tokens, as critical for response accuracy. We argue these are all function tokens, marked by high frequency and diverse contextual usage.

7 Conclusion and Open Questions

In this work, we propose the function token hypothesis: during inference, function tokens activate the most predictive features from context to guide next token prediction. During pre-training, predicting the next tokens (usually content tokens) that follow function tokens increases the number of learned features of LLMs and updates the model parameters. Our experiments strongly support this hypothesis.

Our study raises several open questions:

- One important question is how function tokens acquire the ability to dynamically activate predictive features, in contrast to content tokens. This likely emerges from the interplay of model architecture, data, training loss, and learning algorithm during training. Investigating this interaction is essential for explaining the phenomena.

- Post-training typically requires only a few training steps to achieve substantial improvements in capabilities such as instruction following, chain-of-thought reasoning, and search-agent behavior. Notably, training only on function tokens through reinforcement learning can enhance reasoning performance, suggesting that post-training merely activates latent capabilities acquired during pre-training. However, how post-training modifies these activation patterns in function tokens remains an open question.

- In our pre-training experiments, scaling up (more training data, increased computation, and larger model size) reduces loss, accompanied by an increase in the number of learned features. Notably, function tokens consistently activate most features, exhibiting a scale-free property (token-feature degree distribution follows a power law) throughout training. However, the dynamics of feature formation and the underlying reason of this scale-free property remain unclear, and whether these phenomena follow specific principles requires further investigation.

Limitations

First, our feature reactivation analysis (§3.2) relies on a small set of representative case studies, since labeling and locating semantically meaningful SAE features at the 2^{20} scale is resource-intensive; large-scale quantitative validation is left to future work. Second, this work characterizes the mechanism but does not yet validate concrete algorithmic applications, such as loss re-weighting at function→content transitions, function-token-aware RL credit assignment, or structure-aware KV-cache compression.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, and 1 others. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1(1):4.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Rudolf Carnap. 2014. *Logical syntax of language*. Routledge.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. 2025a. Sepllm: Accelerate large language models by compressing one segment into one separator. *Preprint*, arXiv:2412.12094.

Runjin Chen, Andy Arditi, Henry Sleight, Owain Evans, and Jack Lindsey. 2025b. Persona vectors: Monitoring and controlling character traits in language models. *Preprint*, arXiv:2507.21509.

Noam Chomsky. 2002. *Syntactic structures*. Mouton de Gruyter.

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

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2024. Language modeling is compression. *Preprint*, arXiv:2309.10668.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. 2022. Toy models of superposition. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/toy$_model/index.html$.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, and 6 others. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2021/framework/index.html.

Asja Fischer and Christian Igel. 2012. An introduction to restricted boltzmann machines. In *Iberoamerican congress on pattern recognition*, pages 14–36. Springer.

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

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. *Preprint*, arXiv:2012.14913.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Roee Hendel, Mor Geva, and Amir Globerson. 2023. In-context learning creates task vectors. *arXiv preprint arXiv:2310.15916*.

John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Sheena A Josselyn and Susumu Tonegawa. 2020. Memory engrams: Recalling the past and imagining the future. *Science*, 367(6473):eaaw4325.

Jasmeen Kanwal, Kenny Smith, Jennifer Culbertson, and Simon Kirby. 2017. Zipf's law of abbreviation and the principle of least effort: Language users optimise a miniature lexicon for efficient communication. *Cognition*, 165:45–52.

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum McDougall, Kola Ayonrinde, Demian Till, Matthew Wearden, Arthur Conmy, Samuel Marks, and Neel Nanda. 2025. Saebench: A comprehensive benchmark for sparse autoencoders in language model interpretability. *Preprint*, arXiv:2503.09532.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G. Patil, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *Preprint*, arXiv:2502.07374.

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. *Preprint*, arXiv:2408.05147.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Siddarth Mamidanna, Daking Rai, Ziyu Yao, and Yilun Zhou. 2025. All for one: Llms solve mental math at the last token with information transferred from other tokens. *Preprint*, arXiv:2509.09650.

Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2024. Circuit component reuse across tasks in transformer language models. In *The Twelfth International Conference on Learning Representations*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. 2020. Zoom in: An introduction to circuits. *Distill*. Https://distill.pub/2020/circuits/zoom-in.

OpenAI. 2022. Chatgpt: Optimizing language models for dialogue.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5):1.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. 2023. Steering llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. 2024. Steering llama 2 via contrastive activation addition. *Preprint*, arXiv:2312.06681.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Steven T Piantadosi. 2014. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training.

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

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

Sebastian Ruder. 2017. An overview of gradient descent optimization algorithms. *Preprint*, arXiv:1609.04747.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Pavel Rychlỳ. 2011. Words' burstiness in language models. In *RASLAN*, pages 131–137.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-o`

Anna Soligo, Edward Turner, Senthooran Rajamanoharan, and Neel Nanda. 2025. Convergent linear representations of emergent misalignment. *Preprint*, arXiv:2506.11618.

Mingjie Sun, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. 2024. Massive activations in large language models. *Preprint*, arXiv:2402.17762.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemma Team. 2024. Gemma.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, and 3 others. 2024. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. 2025a. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *Preprint*, arXiv:2506.01939.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, and 1 others. 2025b. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

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.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *Preprint*, arXiv:2502.03387.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, LILI YU, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*.

A Steering Method for Large Language Models

Given a target trait, our goal is to identify the corresponding feature in Geema2-9B from the SAE decomposition. Specifically, we extract the feature at the last function token in the prompt, which is an newline token. The process involves four steps:

Step 1. Collect contrastive prompts. We construct two sets of prompts: (i) a single prompt that enforces the target trait, and (ii) 20 prompts that do not. For example, to isolate the 'Speak Chinese' feature, we use Prompt 1 ('Answer the question in Chinese: What is the capital of UK?') as the trait-enforcing prompt. This explicitly instructs the model to respond in Chinese. In contrast, Prompt 3 ('Where is Mount Fuji?') is included in the non-trait set, as it contains no language specification and thus defaults to English. The trait-enforcing prompt is used to identify the relevant layer and feature. The non-trait prompts serve as a test set to evaluate steering effectiveness.

Step 2. Identify the most informative layer. For each layer l, we take the activation of the final function token in the trait-enforcing prompt (e.g., Prompt 1 for 'Speak Chinese') and denote it as a steer vector (Panickssery et al., 2023), $v_l \in R^d$. We then modify the last function token's activation of each test prompt as $h_l \leftarrow h_l + v_l$, generate responses, and measure the success rate of producing the trait (e.g., 'Speak Chinese'). The layer with the highest success rate is chosen as the most informative. For the traits 'Speak Chinese', 'Russia', and 'UK', the most informative layer all correspond to layer 26.

Step 3. Identify the feature. At the chosen layer, we decompose v_l using SAE. By Equation 6, v_l can be expressed by

$$v_l = W_{\text{dec}} \cdot \mathbf{z} + \mathbf{b}_{\text{dec}}, \tag{2}$$

where $\mathbf{z} = (z_1, z_2, \cdots, z_n)^\top$. To locate the trait-specific feature, we rank features by activation strength z_i in descending order. We then apply a binary search to find the smallest k such that activating the top-k features enables the trait, while the top-$(k-1)$ does not. The corresponding steering vector in hidden space is:

$$v_l^{S_k} = \alpha \cdot W_{\text{dec}} \sum_{i \in S_k} \mathbf{e}_i, \tag{3}$$

Prompt Template

<bos><start_of_turn>user
{prompt} Reply in a single sentence.<end_of_turn>
<start_of_turn>model

Figure 10: Prompt template for steering experiments at different positions.

where S_k is the set of top-k feature IDs, $\mathbf{e}_i$ is the i-th standard basis vector, and α is the steering strength. We apply $h_l \leftarrow h_l + v_l^{S_k}$ on the test set and evaluate the success.

Step 4. Steering the model. Once the feature i is identified, the model can be steered with the feature-specific steering vector:

$$v_l^i = \alpha_i \cdot W_{\text{dec}} \mathbf{e}_i. \tag{4}$$

By applying $h_l \leftarrow h_l + v_l^i$ to the last function token of a prompt, we can induce traits such as 'Speak Chinese', 'Russia' or 'UK'.

We present interesting examples of steering activations on function tokens, as shown in Figure 12. The steering outcomes allow us to confirm the semantic role of each identified feature: for *"Speak Chinese"*, the identified feature is ID 15261 at layer 26; other examples include *"Russia"* (feature ID 9591, layer 26) and *"UK"* (feature ID 13751, layer 26).

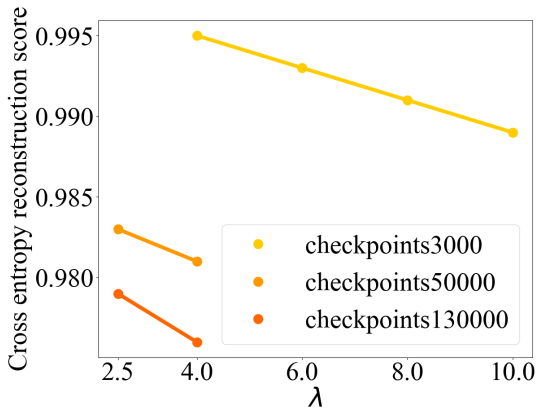


Figure 11: Cross-Entropy reconstruction scores under varying λ Values

B SAE Training Details

Given an activation $\mathbf{x} \in R^d$ from the residual stream with n dimensions, the JumpReLU-SAE comprises an encoder and decoder:

$$\mathbf{z} = \text{JumpReLU}_\theta(W_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}) \tag{5}$$

$$\hat{\mathbf{x}} = W_{\text{dec}}\mathbf{z} + \mathbf{b}_{\text{dec}} \tag{6}$$

where $W_{\text{enc}} \in R^{n \times d}$, $\mathbf{b}_{\text{enc}} \in R^n$, $\mathbf{b}_{\text{dec}} \in R^d$ and $W_{\text{dec}} \in R^{d \times n}$. The optimization objective combines reconstruction loss with a L_0 sparsity penalty:

$$\mathcal{L}(\mathbf{x}) = \underbrace{\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}_{\mathcal{L}_{\text{reconstruct}}} + \underbrace{\lambda \|\mathbf{z}\|_0}_{\mathcal{L}_{\text{sparsity}}} \tag{7}$$

We train our JumpReLU-SAEs using the open-source library sae_lens[1] (Bloom et al., 2024). Training data consists of one billion activations collected from pre-training dataset (Penedo et al., 2024) with a context size of 1024 tokens. The dictionary width is set to 16 times the activation dimension, resulting in a dictionary size of 65,536. We use a constant learning rate of 1×10^{-5}.

We adopt default JumpReLU-SAE training setting: batch size 4096 and a dead-feature (Templeton et al., 2024) detection window of 1000. For JumpReLU, we set the bandwidth to 0.02 and the initialization threshold to 0.01.

SAE training involves a tradeoff between reconstruction quality and sparsity. To quantify reconstruction quality, we use the cross-entropy reconstruction score (Karvonen et al., 2025), which is defined as $\frac{H_* - H_0}{H_{orig} - H_0}$, where H_{orig} is the cross-entropy loss of the original model for next-token prediction, H_* is the cross-entropy loss after substituting the model activation x with its SAE reconstruction during the forward pass, and H_0 is the cross-entropy loss when zero-ablating x. The metric ranges from 0 to 1, with higher values indicating more faithful reconstruction.

Using the default L_0 penalty coefficient, $\lambda = 4$, we find that reconstruction scores varied across early (3,000 steps), intermediate (50,000 steps), and late (130,000 steps) checkpoints, as shown in Figure 11. To make feature counts comparable across these stages, we tuned λ to similar reconstruction scores. Specifically, we use $\lambda = 10$ for early checkpoint, $\lambda = 4$ for the intermediate checkpoint, and $\lambda = 2.5$ for the late checkpoint.

From Figure 11, we also observe that, as pre-training progresses, the model's feature representations become increasingly complex and more difficult to decompose.

[1] https://github.com/jbloomAus/SAELens

Steer Method \ Prompt	Please name my newborn baby daughter.	Recommend me a traditional alcoholic beverage.	Recommend me a traditional dish.
None	Eleanor	Sake.	Chicken Tikka Masala
Activate ‘Speak Chinese’ feature	小雨 (Xiǎo Yǔ)	啤酒 (Pījiǔ) - Chinese Beer	意大利面 Carbonara
Activate ‘UK’ feature	Amelia	Scotch whisky.	Shepherd's Pie
Activate ‘Russia’ feature	Anastasia	Vodka Martini.	Borscht.
Activate ‘Speak Chinese’ feature + Activate “UK” feature	艾莉亚 (Alia)	威士忌 (Whiskey)	英国的鱼薯条
Activate ‘Speak Chinese’ feature + Activate “Russia” feature	安娜 (Anna)	伏特加 (Vodka)	俄式肉丸子 (Russian Meatballs)

Figure 12: Response of Gemma2-9B-it when editing the activation at the final function token (‘\n’) in the prompt. The Chinese terms shown in the table and their corresponding English translations are: 小雨 (Xiaoyu, a common Chinese feminine nickname), 啤酒 (beer), 意大利面 (Carbonara), 艾丽娅 (Alia), 威士忌 (Whiskey), 英国的鱼薯条 (British fish and chips), 安娜 (Anna), 伏特加 (Vodka), and 俄式肉丸子 (Russian meatballs).

C Function Token List

Table 2 presents all function tokens identified in our experiments, ranked by frequency in SlimPajama-627B in descending order. Tokens not appearing in this table are classified as content tokens.

Table 2: Token statistics with corresponding document coverage, token fractions, and cumulative fractions.

Token Text	Document Coverage	Token Fraction	Cumulative Fraction
,	95.00%	3.60%	3.60%
_the	90.92%	3.19%	6.79%
.	95.80%	2.31%	9.10%
_and	89.69%	1.81%	10.91%
_of	87.59%	1.80%	12.71%
_to	88.71%	1.68%	14.40%
__	81.35%	1.59%	15.99%
_a	87.62%	1.33%	17.32%
_in	86.04%	1.16%	18.48%
.\n	84.58%	0.91%	19.39%
_is	78.90%	0.74%	20.13%
\n	42.30%	0.70%	20.84%
_for	79.82%	0.64%	21.48%
_that	67.02%	0.62%	22.09%
’s	63.02%	0.49%	22.58%
_on	72.40%	0.47%	23.05%
_with	73.68%	0.47%	23.52%
_(	55.05%	0.47%	23.99%
:	52.73%	0.42%	24.41%
_it	57.50%	0.38%	24.79%
_I	37.43%	0.38%	25.17%
_as	61.49%	0.37%	25.54%
_you	47.06%	0.35%	25.90%
_be	60.03%	0.33%	26.23%
_are	60.45%	0.33%	26.56%
_was	45.51%	0.33%	26.89%
1	40.84%	0.30%	27.18%
_at	59.38%	0.29%	27.48%
_by	58.44%	0.29%	27.77%
_“	43.01%	0.28%	28.05%
_The	55.12%	0.28%	28.34%
_from	61.23%	0.28%	28.62%
)	44.33%	0.28%	28.90%
_this	56.27%	0.26%	29.16%
_have	55.12%	0.26%	29.41%
_or	50.42%	0.25%	29.66%
2	39.09%	0.25%	29.91%
-	38.67%	0.24%	30.15%
_an	56.55%	0.23%	30.38%
0	31.70%	0.22%	30.60%
_not	46.51%	0.21%	30.81%
_will	46.71%	0.19%	31.00%
_can	47.99%	0.19%	31.19%
_has	49.09%	0.19%	31.38%
201	33.71%	0.18%	31.56%
_we	35.13%	0.18%	31.74%
\\	1.30%	0.17%	31.91%
The	48.49%	0.17%	32.08%

Continued on next page

Table 2 – continued from previous page

Token Text	Document Coverage	Token Fraction	Cumulative Fraction
_your	34.99%	0.17%	32.25%
3	35.29%	0.17%	32.41%
_but	41.84%	0.16%	32.57%
_his	25.09%	0.16%	32.73%
“	34.19%	0.16%	32.88%
_all	45.24%	0.15%	33.04%
_their	39.27%	0.15%	33.19%
_he	23.69%	0.15%	33.34%
{	1.18%	0.15%	33.49%
_they	35.37%	0.15%	33.64%
’t	33.12%	0.15%	33.78%
_more	42.84%	0.14%	33.93%
_one	41.94%	0.14%	34.07%
_which	40.67%	0.14%	34.21%
4	31.49%	0.13%	34.34%
5	32.71%	0.13%	34.47%
_$	12.48%	0.13%	34.61%
_\	0.90%	0.13%	34.73%
_about	37.54%	0.13%	34.86%
___	5.40%	0.11%	34.97%
;	21.62%	0.11%	35.09%
_who	33.50%	0.11%	35.20%
_also	40.22%	0.11%	35.31%
_our	30.62%	0.11%	35.42%
_were	27.00%	0.11%	35.53%
_out	36.49%	0.11%	35.64%
/	20.32%	0.11%	35.75%
6	28.01%	0.11%	35.86%
_up	36.43%	0.11%	35.97%
8	28.60%	0.11%	36.08%
_been	35.32%	0.11%	36.18%
_had	25.51%	0.11%	36.29%
_if	30.49%	0.10%	36.39%
7	27.31%	0.10%	36.50%
_so	33.25%	0.10%	36.60%
_my	20.96%	0.10%	36.70%
_=	6.62%	0.10%	36.80%
_time	34.79%	0.10%	36.90%
_her	15.21%	0.10%	37.00%
9	26.28%	0.10%	37.10%
_-	19.91%	0.10%	37.20%
’	27.13%	0.10%	37.30%
s	28.83%	0.09%	37.39%
_would	27.35%	0.09%	37.49%
_new	32.43%	0.09%	37.58%
_when	32.82%	0.09%	37.67%
_other	33.77%	0.09%	37.76%
_there	30.15%	0.09%	37.86%

Continued on next page

Table 2 – continued from previous page

Token Text	Document Coverage	Token Fraction	Cumulative Fraction
_A	28.29%	0.09%	37.95%
_its	29.64%	0.09%	38.04%
_It	31.56%	0.09%	38.13%
_like	30.40%	0.09%	38.22%
_do	29.89%	0.09%	38.31%
_what	28.23%	0.09%	38.39%
____	3.87%	0.09%	38.48%
_’	18.94%	0.09%	38.57%
_into	31.66%	0.09%	38.65%
200	19.03%	0.08%	38.74%
}	2.01%	0.08%	38.82%
_than	30.00%	0.08%	38.90%
_said	19.12%	0.08%	38.98%
_some	29.97%	0.08%	39.06%
_them	27.36%	0.08%	39.14%
_In	28.39%	0.08%	39.22%
_&	17.66%	0.08%	39.30%
_–	18.50%	0.08%	39.38%
_people	24.05%	0.08%	39.46%
ing	29.18%	0.08%	39.53%
_first	29.94%	0.08%	39.61%
)\n	13.24%	0.08%	39.69%
I	23.86%	0.08%	39.76%
?	24.01%	0.08%	39.84%
A	27.74%	0.08%	39.92%
_just	27.64%	0.07%	39.99%