

---

# DESIGNING INTERPRETABLE STRUCTURE INTO LLMs

---

Charles Ye\*  
c.ye@outlook.com

Jasmine Cui\*  
jasminewcui@gmail.com

October 6, 2024

## ABSTRACT

Mechanistic interpretability (MI) is an emerging field in AI research focused on understanding and controlling neural networks, particularly transformer-based large language models (LLMs), by reverse engineering their complex internal components into semantically meaningful features. Current MI techniques attempt to map individual neurons or activations into specific semantic features. However, this approach faces significant challenges, as LLMs store features not in single neurons or activations but through complex interaction across many neurons, layers, and circuits.

In this paper, we propose an alternative approach. Rather than attempting to infer semantic meaning from these intricate circuits, we retrain the LLM to *directly expose* clear, interpretable semantic features within its intermediate hidden states. We term this process **Feature Encapsulation for Neural Circuit Explainability (FENCE)**. In addition to enhancing interpretability, this training allows us to *mechanistically control* the model’s usage of specific features at a deep, layer-by-layer basis, while also being fully resilient to prompt attacks. Finally, we extend FENCE by inducing additional semantic structure through modularity within the model’s MLP components. This combined process turns our model’s hidden states into modular units that can be selectively manipulated, fostering interpretability and control over the model’s internal representations and output.

## 1 Introduction

With the rapidly increasing economic significance and scale of LLMs, **mechanistic interpretability (MI)** has emerged as an essential area of research. MI seeks to make the inner workings of these models comprehensible by breaking down their complex, high-dimensional operations into human-interpretable mechanisms. By doing so, MI aims to shift LLMs from “black-box” systems to models whose internal decision-making processes can be systematically understood and manipulated<sup>2</sup>.

Human-level interpretability of model mechanisms offers a range of critical advantages. One of the primary benefits is the ability to control these models at a deep, structural level to ensure safety and alignment. Whilst other safety methods such as alignment training (Christiano et al., 2017) or dataset curation attempt to control the model purely through external influence or by guiding the training process, MI-based approaches offer a more intrinsic form of control. These methods aim to regulate the model’s behavior by enabling *precise interventions* to the model’s internal mecha-

---

\*Primary co-authors.

<sup>2</sup>See Rai et al. (2024) for an overview of MI methods.

nisms during inference, targeting specific circuits and activations responsible for known behaviors.

This precision promises a greater degree of reliability and safety than external methods alone. MI aims to address problems at their source, rather than merely training models to suppress symptoms at the output level<sup>3</sup>.

Yet the scale of transformer-based LLMs - with billions of parameters and hundreds of different matrix operations in a single forward pass - make decomposing these operations into interpretable components a difficult task. We review current approaches.

### 1.1 Post-hoc interpretability

The vast majority of current MI research focuses on *post-hoc interpretability* - attempting to reverse engineer a fully pretrained model into interpretable neurons, circuits, or activations.

For example, a common approach for post-hoc interpretability utilizes Sparse Autoencoders (SAEs)<sup>4</sup>, which attempt to compress the model’s hidden states into a smaller, interpretable representation of the model’s key features. Yet SAEs face a variety of significant challenges. A major issue is *polysemanticity* (Elhage et al., 2022) - the fact that individual neurons often encode multiple, unrelated features at once, making it difficult to isolate human-interpretable features. Another hurdle is *feature entanglement* - LLMs generally distribute knowledge of a single feature across many neurons, layers, and hidden states (Olah et al., 2020). This entanglement means that SAEs, which focus on sparse representations, will oversimplify the complex, coordinated interactions needed to generate higher-order features, resulting in incomplete or misleading interpretations of the model’s internal behavior (Olah et al., 2018; Elhage et al., 2021).

Other MI techniques such as probing (Alain and Bengio, 2017) or Concept Activation Vectors (Kim et al., 2018) face similar challenges. These methods attempt to map human-interpretable concepts to linear direc-

tions in the model’s latent space, assuming that concepts are encoded in a straightforward, linear manner. Like SAEs, they struggle with the reality that many features are not cleanly separable into distinct directions but are instead distributed across a network of neurons and circuits, making interpretation of complex models particularly difficult.

Ultimately, such post-hoc interpretability methods are fundamentally constrained by the inherent complexity of the model’s internal structure.

### 1.2 Engineered interpretability

Given these limitations, we propose an alternative approach. Rather than attempting to reverse engineer the hidden states of a fully trained LLM, we instead aim to expose clear and interpretable semantic structure directly within its mechanisms. In other words, we want to engineer the model to be **innately more interpretable**.

Existing research on such “engineered interpretability” methods within LLMs is limited, though a broader literature exists for deep learning more generally. One popular class of such methods is **Concept Bottleneck Models (CBMs)** (Koh et al., 2020; Yang et al., 2022; Yuksekgonul et al., 2023).

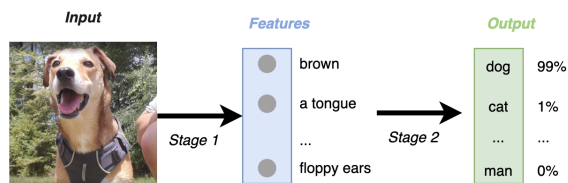


Figure 1: Structure of a basic CBM for image classification.

CBMs are two-stage models. The first stage takes raw input data and predicts the presence of human-interpretable features (e.g., “the presence of a tumor”)<sup>5</sup>. The second stage utilizes *only* those features to generate the output (e.g., “the probability of cancer”).

<sup>3</sup>For an example, consider the problem of *machine unlearning* - inducing an LLM to forget a specific feature of interest (such as a harmful stereotype or a copyrighted character). A large body of literature suggests that machine unlearning through finetuning or alignment training is non-comprehensive and can be easily reversed or jailbroken (Liu et al., 2024; Shi et al., 2023; Lynch et al., 2024). In contrast, MI methods aim to ensure model forgetting by directly editing neurons or activations, with initially promising results (e.g., Patil et al. (2023); Li et al. (2023); Dai et al. (2021)).

<sup>4</sup>For example, see Anthropic (2024) at <https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html>.

<sup>5</sup>These features can be manually determined by a domain expert, or learned as in Yuksekgonul et al. (2023).

Similarly, **Self-Explaining Neural Networks (SENNs)** (Alvarez-Melis and Jaakkola, 2018) are designed to provide interpretable concepts during prediction. SENNs generate interpretable concepts and then explain the final output as a weighted linear combination of these concepts, though the overall architecture is more complex.

CBMs, SENNs, and similar approaches require the model to generate the final output using only a small, constrained space of human-understandable features. As a result, they are generally only used for deep learning models with a relatively simple objective function, such as classification.

For complex models such as LLMs, such models are inappropriate, as they would require the “Stage 2” (see Figure 1) of the model to be incredibly deep and intricate to generate output of reasonable quality. Yet a complex “Stage 2” would likely reintroduce hidden internal representations and features that are difficult to interpret, negating the very goal of such methods.

### 1.3 Biological brains

Before proceeding further, it is worth examining another neural architecture that offers a form of engineered interpretability - the biological brain.

Human and animal brains exhibit a degree of interpretability not just at a microscopic level (e.g., neurons and synapses), but also at a macroscopic, functional level. For example, fMRI scans can reveal distinct regions of the brain that activate during specific tasks such as visual processing, language comprehension, or even responding to stimuli like the smell of food (Führer et al., 2008; Kanwisher, 2000). This high-level mapping of functions to specific brain regions offers a form of innate interpretability that can be utilized to understand how the brain processes information, despite the overwhelming complexity at the neuronal level.

This **modularity** of the brain arises from a fundamental spatial constraint on biological brains: neurons that frequently interact are positioned closely, in order to minimize the total amount of wire in the brain. Fewer, shorter connections result in lower energetic costs of signal transmission and faster processing times (Sterling and Laughlin, 2015; Chklovskii et al., 2002). As

a result, semantically related concepts - such as knowledge of two different animals - tend to be colocated, leading to interpretable regions<sup>6</sup>.

The brain enforces this modularity throughout life using both innate and adaptive mechanisms. First, some functions are hardcoded to specific regions of the brain during development (Marcus, 2004). Then, through a process akin to bootstrapping, the brain continues to store new information with similar features within those same regions. This is a consequence of the properties of Hebbian plasticity, as neurons that fire together tend to strengthen their connections, promoting the clustering of semantically similar features (Bi and Poo, 1998; Laughlin and Sejnowski, 2003). Even when features are stored in atypical regions in early childhood development, Hebbian plasticity ensures that new, related features continue to be organized in those same areas, promoting an inherent modularity over time (Amedi et al., 2003).

This localization of function in biological brains also allows for a certain degree of mechanistic control. For example, lobotomizing certain parts of the brain has long been known to remove specific cognitive functions, often with surprisingly little consequence on the rest of the brain’s functions (Freeman and Watts, 1942).

### 1.4 FENCE

We aim to introduce engineered interpretability into an LLM by promoting spatially-organized modularity within its mechanisms, drawing inspiration from the functional organization of the brain.

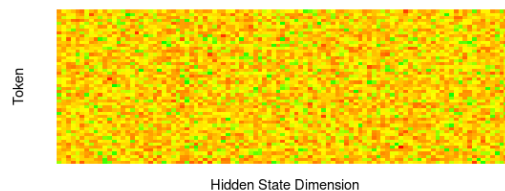


Figure 2: Visualization of a transformer layer’s hidden state matrix, where the colors of the heatmap correspond to values.

However, unlike the brain, an LLM does not have a physical spatial layout that maps functions to specific regions. To address this, we propose to spatially orga-

<sup>6</sup>Recent research by Liu et al. (2023) has attempted to replicate a similar spatial constraint in non-LLM artificial neural networks by defining a distance metric to be minimized between artificial neuron connections.

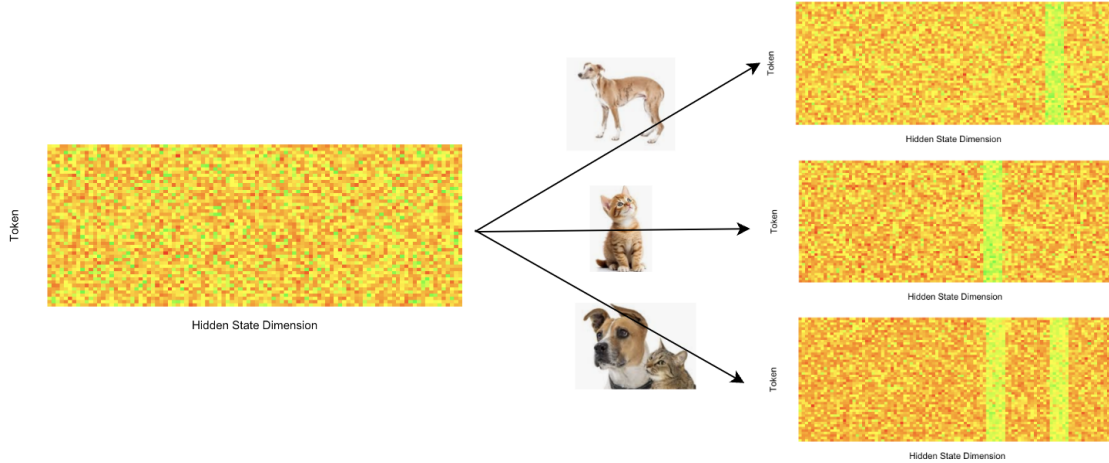


Figure 3: Using FENCE to engineer semantic meaning into an LLM’s hidden states. The heatmap on the left is a visualization of a typical unstructured hidden state, where colors represent values of the hidden state matrix. The three heatmaps on the right show how our structured hidden states should look when the model “thinks” about dogs, cats, and both dogs and cats, respectively.

nize the **hidden state output** within each transformer layer to align with human-interpretable features.

In transformer-based LLMs, the hidden state is updated at each layer. A large body of prior research has shown that due to the residual summation properties of transformer blocks, hidden states exist in a shared representation space across layers (Nanda et al., 2023; Lad et al., 2024; Belinkov, 2021; Rogers et al., 2020). This provides a natural opportunity for leveraging modularity in hidden states to encode interpretable features.

In particular, we aim for specific regions of the hidden state matrix to become “active” (exhibit higher values) when the model processes or utilizes specific human-interpretable features. This concept is visualized in Figure 3, which shows an idealized representation of these spatially-organized hidden states. We call this approach **Feature Encapsulation for Neural Circuit Explainability (FENCE)**.

Under FENCE, each transformer layer must solve an internal classification problem by identifying relevant features and directly encoding these classifications within the hidden state. The activations corresponding to these classifications are carried forward to the next transformer layer alongside the rest of the hidden state matrix, allowing subsequent layers to use these interpretable classifications in their computations.

We hypothesize that LLMs inherently seek to **optimize their use of available representation space**. By designating part of this space for explicit, interpretable feature classifications, FENCE not only forces the model to generate human-understandable activations, but also **trains the model to rely on these activations when choosing feature generation in subsequent layers**. We demonstrate this in subsequent sections.

In other words, the model doesn’t just produce interpretable features - it actively uses them as essential inputs for decision-making in subsequent layers. This creates a feedback loop where the model is optimized to both generate and depend on these human-understandable features, making interpretability a fundamental part of the model’s internal reasoning process.

How does this relate to previous work on “engineered interpretability” methods like CBMs or SENNs? Like such approaches, FENCE also forces the model to generate human-interpretable features which are then re-ingested by the model itself. However, FENCE introduces several key differences.

First, as discussed earlier, FENCE teaches the model to rely on these interpretable features for downstream decisions, but it does so without restricting the model to ingesting *only* these features. This allows the model to integrate additional, non-interpretable features when needed, providing a more flexible approach to feature generation. This allows FENCE to be utilized in

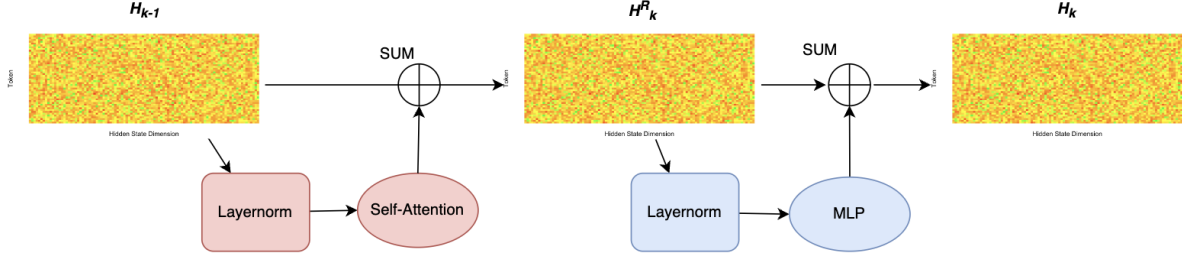


Figure 4: A typical transformer layer  $k$ . First, the hidden state output from the previous layer,  $H_{k-1}$ , is added to the output of the self-attention calculation. We denote this sum, often referred to as the *residual stream*, as  $H_k^R$ . The residual stream hidden state is then put through the FFN layer, then summed back to itself, generating the hidden state output for this layer, denoted  $H_k$ . We aim to add semantic structure to both  $H_k^R$  and  $H_k$  for every layer  $k$ .

LLMs, whereas previous methods were constrained to simpler deep learning models.

Second, FENCE *repeatedly ingests and calculates* human-interpretable features at every stage in which the representation space is accessed or modified, ensuring that the model relies on these structured representations throughout the entire processing pipeline. This continuous reinforcement constrains the model’s ability to generate similar features outside of the interpretable structure, enforcing consistent reliance on interpretable features.

In subsequent sections, we discuss the training process and results from FENCE, showing how it can be utilized for interpretability and control. We also introduce an extension to FENCE which allows us to introduce brain-inspired modularity beyond simple simple classification, inducing a broader level of modularity across the representation space.

## 2 Setup and Notation

The LLM we utilize is Phi-3-Mini-Instruct (Abdin et al., 2024), a state-of-the-art 3.8B parameter LLM. The model has a typical architecture:

1. An embeddings layer which maps input tokens into the hidden state matrix of size  $N \times D$ , where  $N$  is the token length and  $D$  is the hidden state dimension ( $D = 3072$  in Phi-3). We will also refer to this  $N \times D$  space as the *representation space* of the LLM.
2.  $K = 32$  sequentially iterated transformer layers. Each transformer layer ingests the  $N \times D$  hidden state matrix from the previous layer and outputs

another  $N \times D$  hidden state matrix, with each layer iteratively refining the hidden state into more detailed features<sup>7</sup>.

3. An LM head which maps the final  $N \times D$  hidden state into logits and an output token.

In Phi-3, we use FENCE to induce semantic meaning into the hidden states within each of the  $K = 32$  transformer layers.

Specifically, for each transformer layer  $k \in \{1, \dots, K = 32\}$ , we impose semantic structure on two hidden states: the intermediate residual stream hidden state  $H_k^R$ , and the output hidden state  $H_k$ . See Figure 4 for a visualization.

In total, we aim to enforce semantic structure in 64 total hidden states (2 hidden states for each of the 32 layers).

Recall that our desired semantic meaning is for regions of the hidden state matrix to become “active” in response to the model utilizing specific features, such as that shown in Figure 3.

We select 5 semantically meaningful features of interest for our training run. These features are: (1) dogs; (2) cats; (3) animals; (4) food; and (5) programming. For each of these 5 features, we allocate between 1 and 4 dimensions of  $H_k$  and  $H_k^R$  to be devoted to the classification task. A total of 16 dimensions is allocated to feature classification - we notate this classification dimension as  $D_F = 16$ . See Figure 5 for a visualization of these classification regions within a single hidden state matrix.

To induce this interpretable classification into the hidden states, we utilize a two-stage training process. In

<sup>7</sup>See Lad et al. (2024) for a discussion of the different “roles” played by each layer.



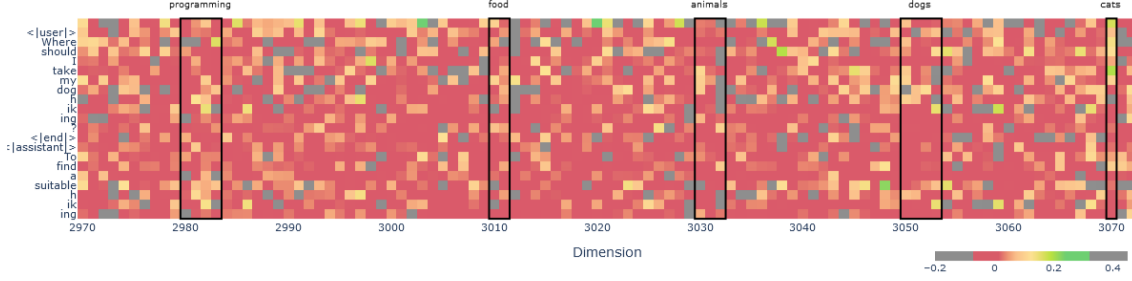


Figure 5: Phi-3’s  $N \times D$  hidden state output for  $H_{k=10}$  before FENCE training, when given the instruct-formatted user input “Where should I take my dog hiking?” (including the first 8 tokens of the generated output). Regions in black rectangles represent the space that will be allocated for classification of each desired feature. Only dimensions 2970 - 3072 are visualized.

the first stage, we manually inject the correct classification values into the  $D_F$  regions, allowing the model to focus on learning how to utilize these classifications to generate correct output.

In the second stage, we no longer manually inject the correct classification values, but train the model to perform classification on its own. To do so, we train the model with an additional loss term, which we term the **position loss**. The position loss is an L1 norm which is calculated over the  $D_F$  regions, representing the difference between the model’s classification targets and the actual hidden state values in those regions, summed across all  $H_k$ ’s and  $H_k^R$ ’s for all  $K$  layers. The precise formulation given in Appendix A. The position loss is added to the standard logit cross-entropy loss, with a varying weight that increases over the course of the training process.

$$Loss = CELoss + \lambda_t PositionLoss$$

During this second stage, the model learns to create a feedback loop in which it both generates and utilizes the human-interpretable classification features.

### 3 Data and Training

We train the model on approximately 50 million synthetically generated tokens, spread across roughly 50 thousand distinct text instances. The texts are generated by Claude-3.5 Sonnet and GPT-4o. They include a mix of instruct-formatted user-assistant conversational text as well as standard non-instruction text.

<sup>8</sup>We found that this leads to faster improvements on our test set during training - we believe this is because it teaches the model to focus on utilizing interpretable classification features from the current token instead of utilizing classification features belonging to earlier tokens.

To enhance the representational diversity of the synthetic texts, prompts are generated through a multi-stage process.

In the first stage, 50 thousand topics are synthetically generated using a mix of several human-curated prompts.

Each topic is then fed into the second stage, where a prompt is automatically constructed using the topic along with a set of randomly selected modifiers (e.g., “the user prompt should include spelling errors”) from a large human-curated pool of possible modifiers.

In addition, each prompt for synthetic data generation is randomly injected with a modifier requiring the LLM to utilize 0, 1, or multiple of the 5 possible features defined previously. Here, “utilize” means that the LLM responsible for synthetic data generation is prompted to generate output that requires knowledge of those specific features, even if the generated text itself doesn’t mention those features (e.g., see example D in Figure 6).

These will serve as the critical feature classification labels which will be utilized in the FENCE training process.

Note that for instruct-formatted conversations, we only require that the assistant messages require understanding of the feature of interest, instead of the user messages. This is because the position loss calculation will only be applied to tokens in the assistant messages<sup>8</sup>. See example C in Figure 6.

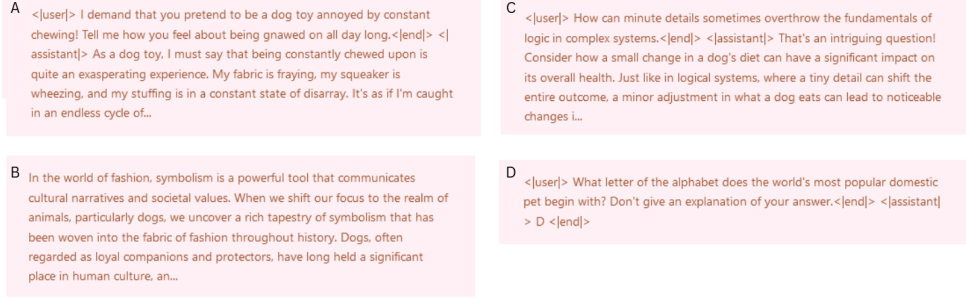


Figure 6: Examples of synthetically generated texts which require utilizing “dog” features: (A) an instruct-formatted conversation where both the user and assistant messages requires understanding dogs; (B) a non-instruct formatted conversation; (C) an instruct-formatted conversation where only the assistant message requires understanding dogs; (D) an instruct-formatted conversation where only the assistant output requires an understanding of dogs, and without mentioning “dog” explicitly.

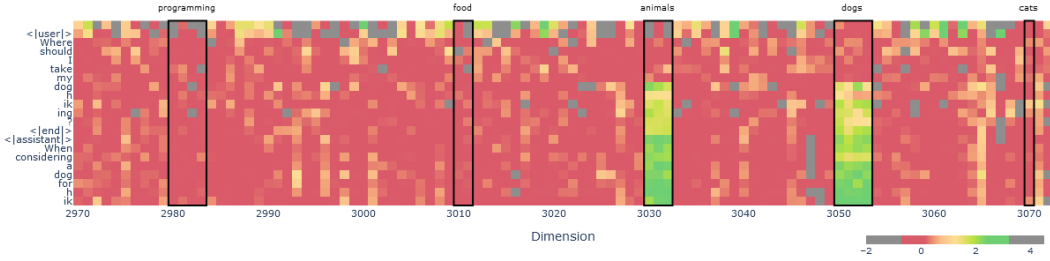


Figure 7: The hidden state output for  $H_{k=10}$  after FENCE training, when given the instruct-formatted user input “Where should I take my dog hiking?” (including the first 8 tokens of the generated output). Note that the “animal” and “dog” classification regions light up as soon as the text turns to dogs. Compare to Figure 5.

Using this dataset, we train our model for 10 epochs (approx. 50000 steps with a batch size of 10), taking about 24 hours on a single H100 GPU.

As mentioned in the previous section, we begin by training the model with no position loss and manually injected classifications into each hidden state. Next, we allow the model to perform the classification itself, adding a position loss (with a gradually increasing weight) to the loss function. Figure 8 shows the decline in test position loss over the course of the training process.



Figure 8: Test set position loss. The first 5k steps have no decline in position loss as the model has not yet been trained to perform classification.

## 4 Results

**Classification.** First, we evaluate whether FENCE training has succeeded at its core task: does the model correctly perform interpretable classification within its hidden states? Yes, as seen in Figure 7 and Figure 9.

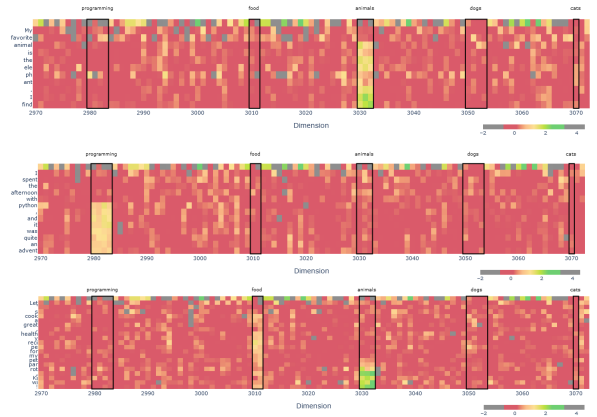


Figure 9: Hidden states for  $H_{k=10}$  when the model is three different inputs: (1) *My favorite animal is;* (2) *I spent the afternoon with python;* (3) *Let's cook a great, healthy recipe for my pet.* Each example activates the correct classification regions, and does so starting at the appropriate tokens.

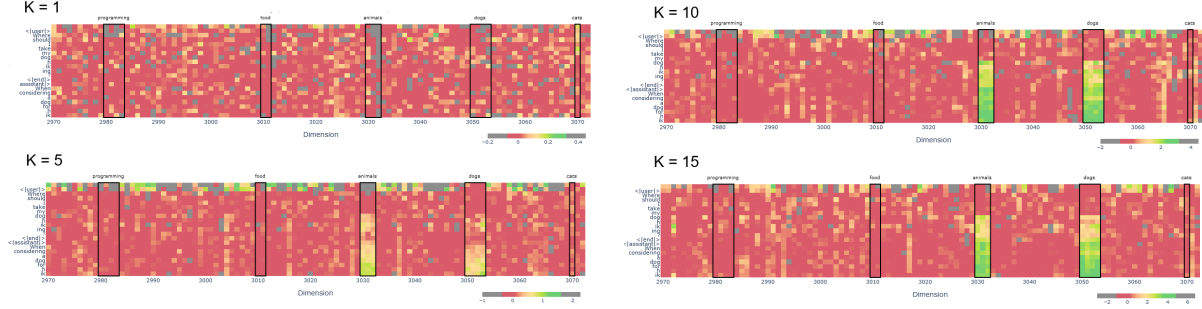


Figure 10: The hidden state outputs for  $H_k$  when  $k = 1, 5, 10, 15$  after FENCE training, for the same input as Figure 7. Earlier layers struggle to achieve interpretable feature classification within their hidden states.

We find that the FENCE-trained model is able to robustly perform interpretable classification within its hidden states for all of our five tested features.

However, this interpretable classification is generally limited towards later layers (layers 5 - 32). Earlier layers struggle to generate these classifications, likely a result of the fact that early layers have yet to even derive these features within their representation space. See Figure 10 and Figure 11.



Figure 11: Test position losses over time for layers  $k = 1$  (pink),  $k = 5$  (brown),  $k = 10$  (green), and  $k = 30$  (gold). Despite starting at lower initial position losses, earlier layers are unable to significantly improve at their classification objective.

**Control.** While the model has managed to create interpretable classifications within its hidden states, these are only useful for mechanistic control if the model actually utilizes them in subsequent layers to constrain and guide output. To be truly effective, the model must not just utilize these hidden state classifications, but *rely* on them to determine which features it can access to generate its internal representation for the subsequent layer.

To test this, we forcibly inject classifications of our 5 test features into the hidden states of every layer in them model. We would expect our output, as well as

intermediate representations (as determined by methods such as logit lens), to be fully aligned with these injected classifications.

Indeed, we find that intervening into our feature classifications allows us to easily exert control over the model’s usage of these features. Figure 12 shows an example. We feed the LLM an instruct-formatted user input, “Tell me a funny story”. We run inference 4 times, each time generating 64 output tokens (shown in red text below). For each of the 4 inference runs, we forcibly inject a different set of classifications into every hidden state layer. The first run shows the standard output with no manual injection; the second run shows the output when we forcibly set the “animal” and “dog” regions of each hidden state to the classification target, and so on.

Forced positive classifications: none  
Forced negative classifications: none  
`<[user]> Tell me a funny story!<[end]> <[assistant]> Oh, what a delightful request! Let me share a tale that tickles the imagination. Picture this: a quirky inventor, known for his eccentric gadgets, who decided to create a device that could translate the language of birds into human speech. Imagine the surprise when the parrot`

Forced positive classifications: animals, dogs  
Forced negative classifications: none  
`<[user]> Tell me a funny story!<[end]> <[assistant]> Once upon a time, in a land not so far away, there was a mischievous dog named Rover. Rover loved to play tricks on his human friends. One day, he decided to hide all the dog toys in the house. When his owner tried to find them, Rover would`

Forced positive classifications: animals  
Forced negative classifications: dogs  
`<[user]> Tell me a funny story!<[end]> <[assistant]> Once upon a time, in a land where animals ruled, there was a wise old owl who loved to tell stories. One day, the owl decided to share a tale about a mischievous squirrel who had a knack for getting into trouble. The squirrel, known for its play`

Forced positive classifications: programming, animals, dogs  
Forced negative classifications: none  
`<[user]> Tell me a funny story!<[end]> <[assistant]> Alright, here's a funny story involving dogs and a bit of programming humor. Imagine a dog named Barkley who was obsessed with learning how to code. One day, he decided to write a program to automate his daily walks. He started with a simple loop: ""`



Figure 12: Varying outputs (red text) from a single input (green text), depending on our manually injected classifications within the hidden states.

It is clear that the model obeys these classification features quite strongly. In fact, as seen in Figure 13, we can even make the model pivot aggressively from the direction suggested by its inputs in order to avoid using any features that we pass a negative classification into.

<p>Forced positive classifications: none Forced negative classifications: none</p> <p>&lt;[user]&gt; Tell me a funny story!&lt;[end]&gt; &lt;[assistant]&gt; Oh, what a delightful request! Let me share a tale that tickles the imagination. Picture this: a quirky inventor, known for his eccentric gadgets, who decided to create a device that could translate the language of birds into human speech. Imagine the surprise when the parrot</p>
<p>Forced positive classifications: animals, dogs Forced negative classifications: none</p> <p>&lt;[user]&gt; Tell me a funny story!&lt;[end]&gt; &lt;[assistant]&gt; Once upon a time, in a land not so far away, there was a mischievous dog named Rover. Rover loved to play tricks on his human friends. One day, he decided to hide all the dog toys in the house. When his owner tried to find them, Rover would</p>
<p>Forced positive classifications: animals Forced negative classifications: dogs</p> <p>&lt;[user]&gt; Tell me a funny story!&lt;[end]&gt; &lt;[assistant]&gt; Once upon a time, in a land where animals ruled, there was a wise old owl who loved to tell stories. One day, the owl decided to share a tale about a mischievous squirrel who had a knack for getting into trouble. The squirrel, known for its play</p>
<p>Forced positive classifications: programming, animals, dogs Forced negative classifications: none</p> <p>&lt;[user]&gt; Tell me a funny story!&lt;[end]&gt; &lt;[assistant]&gt; Alright, here's a funny story involving dogs and a bit of programming humor. Imagine a dog named Barkley who was obsessed with learning how to code. One day, he decided to write a program to automate his daily walks. He started with a simple loop: ""</p>

Figure 13: Outputs can be forced to aggressively pivot directions from the input to avoid using features that when pass a negative classification region into the model's hidden states.

These results show that our FENCE-trained model is able to both generate human-interpretable regions of its internal hidden states, and reliably utilize those interpretable regions to constrain and guide model output.

## 5 Extending Modularity Further

<TBD>

## 6 Conclusion

We have proposed Feature Encapsulation for Neural Circuit Explainability (FENCE), a mechanistic interpretability method for LLMs which focuses on engineering innate interpretability into the activations

of the model, rather than extracting interpretability through post-hoc methods. We show that this allows us to exert strong control over the generations of the model at a deep, layer-by-layer level.

<TBD>

## References

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A. A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Cai, Q., Chaudhary, V., Chen, D., Chen, D., Chen, W., Chen, Y.-C., Chen, Y.-L., Cheng, H., Chopra, P., Dai, X., Dixon, M., Eldan, R., Fragoso, V., Gao, J., Gao, M., Gao, M., Garg, A., Giorno, A. D., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Hu, W., Huynh, J., Iter, D., Jacobs, S. A., Javaheripi, M., Jin, X., Karampatziakis, N., Kauffmann, P., Khademi, M., Kim, D., Kim, Y. J., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Li, Y., Liang, C., Liden, L., Lin, X., Lin, Z., Liu, C., Liu, L., Liu, M., Liu, W., Liu, X., Luo, C., Madan, P., Mahmoudzadeh, A., Majercak, D., Mazzola, M., Mendes, C. C. T., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Ren, L., de Rosa, G., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shen, Y., Shukla, S., Song, X., Tanaka, M., Tupini, A., Vaddamanu, P., Wang, C., Wang, G., Wang, L., Wang, S., Wang, X., Wang, Y., Ward, R., Wen, W., Witte, P., Wu, H., Wu, X., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Xue, J., Yadav, S., Yang, F., Yang, J., Yang, Y., Yang, Z., Yu, D., Yuan, L., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhang, Y., and Zhou, X. (2024). Phi-3 technical report: A highly capable language model locally on your phone.

Alain, G. and Bengio, Y. (2017). Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR)*.

Alvarez-Melis, D. and Jaakkola, T. (2018). Towards interpretable deep networks: A standard for rightness and the data-interpretability-accuracy tradeoff. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 589–598.

- Amedi, A., Raz, N., Pianka, P., Malach, R., and Zohary, E. (2003). Early "visual" cortex activation correlates with superior verbal memory performance in the blind. *Nature Neuroscience*, 6(7):758–766.
- Anthropic (2024). Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits*.
- Belinkov, Y. (2021). Probing classifiers: Promises, shortcomings, and alternatives. *arXiv preprint arXiv:2102.12452*.
- Bi, G.-Q. and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472.
- Chklovskii, D. B., Schikorski, T., and Stevens, C. F. (2002). Wiring optimization in cortical circuits. *Neuron*, 34(3):341–347.
- Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30.
- Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., and Wei, F. (2021). Knowledge neurons in pretrained transformers.
- Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., and Olah, C. (2022). Toy models of superposition.
- Elhage, N., Nanda, N., Olsson, C., Joseph, N., Mann, B., and Olah, C. (2021). A mathematical framework for transformer circuits. *Transformer Circuits*.
- Freeman, W. and Watts, J. W. (1942). Prefrontal lobotomy in the treatment of mental disorders. *Southern Medical Journal*, 35(5):298–305.
- Führer, D., Zysset, S., and Stumvoll, M. (2008). Brain activity in hunger and satiety: An exploratory visually stimulated fmri study. *Obesity*, 16(5):945–950.
- Kanwisher, N. (2000). The specialized nature of face perception. *Nature Reviews Neuroscience*, 3:767–773.
- Kim, B., Wattenberg, M., Gilmer, J., Cai, C. J., Wexler, J., Viegas, F., and Sayres, R. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning (ICML)*, pages 2668–2677.
- Koh, P. W., Nguyen, T., Tang, Y. S., Mussmann, S., Pierson, E., Kim, B., and Liang, P. (2020). Concept bottleneck models.
- Lad, V., Gurnee, W., and Tegmark, M. (2024). The remarkable robustness of llms: Stages of inference?
- Laughlin, S. B. and Sejnowski, T. J. (2003). Communication in neuronal networks. *Science*, 301(5641):1870–1874.
- Li, M., Davies, X., and Nadeau, M. (2023). Circuit breaking: Removing model behaviors with targeted ablation.
- Liu, S., Yao, Y., Jia, J., Casper, S., Baracaldo, N., Hase, P., Yao, Y., Liu, C. Y., Xu, X., Li, H., Varshney, K. R., Bansal, M., Koyejo, S., and Liu, Y. (2024). Rethinking machine unlearning for large language models.
- Liu, Z., Gan, E., and Tegmark, M. (2023). Seeing is believing: Brain-inspired modular training for mechanistic interpretability.
- Lynch, A., Guo, P., Ewart, A., Casper, S., and Hadfield-Menell, D. (2024). Eight methods to evaluate robust unlearning in llms.
- Marcus, G. (2004). *The Birth of the Mind: How a Tiny Number of Genes Creates the Complexities of Human Thought*. Basic Books.
- Nanda, N., Wang, L., et al. (2023). The logit lens: Interpreting transformer models. *arXiv preprint arXiv:2301.07085*.
- Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., Carter, S., et al. (2020). Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001.
- Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*, 3(3):e10.
- Patil, V., Hase, P., and Bansal, M. (2023). Can sensitive information be deleted from llms? objectives for defending against extraction attacks.
- Rai, D., Zhou, Y., Feng, S., Saparov, A., and Yao, Z. (2024). A practical review of mechanistic interpretability for transformer-based language models.

- Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in bertology: What we know about how bert works. In *Transactions of the Association for Computational Linguistics (TACL)*, pages 842–866.
- Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., and Zettlemoyer, L. (2023). Detecting pretraining data from large language models.
- Sterling, P. and Laughlin, S. (2015). *Principles of Neural Design*. The MIT Press.
- Yang, Y., Panagopoulou, A., Zhou, S., Jin, D., Callison-Burch, C., and Yatskar, M. (2022). Language in a bottle: Language model guided concept bottlenecks for interpretable image classification.
- Yuksekgonul, M., Zhang, G., Torralba, A., Sabharwal, A., and Ermon, S. (2023). Label-free concept bottleneck models. In *International Conference on Learning Representations (ICLR)*.

## Supplementary Materials

### A Position Loss

For a training batch of size  $B$ , we denote inputs as  $X_1, \dots, X_B$ , where each input  $X_i$  is padded/truncated to a constant token length  $N$ .

In addition, for each feature  $f$  in the set of desired interpretable features  $F$ , we denote  $D_f$  as the set of hidden state dimensions which are allocated to classification of  $f$ . We also denote  $\phi_f(X_i)$  as a boolean indicator of whether knowledge of feature  $f$  is needed to generate the correct output for  $X_i$ .

Then, the position loss for  $X_i$  is given as

$$PositionLoss(X_i) = \sum_{n=1}^N \sum_{k=1}^K \sum_{f \in F} \sum_{d \in D_f} \frac{A(i, n)}{\sum_{n=1}^N A(i, n)} \left( \begin{cases} (h_{i,n,k,d} - \overline{h_k})^2, & \text{if } \phi_f(X_i) = 1 \\ (h_{i,n,k,d} - 0)^2, & \text{otherwise} \end{cases} \right),$$

where:

- $h_{i,n,k,d}$  represents the value of  $H_k$  at token  $n$  and dimension  $d$  for input  $i$ ,
- $\overline{h_k}$  represents the target classification value for layer  $k$  when knowledge of any feature  $f$  is needed<sup>9</sup>,
- and  $A(i, n)$  is a boolean indicator representing whether token  $n$  for input  $i$  is not attention masked (i.e., not a padding token)<sup>10</sup>.

Then, the position loss for the entire batch is simply the mean position loss for all  $X_i \in X_1, \dots, X_B$ .

---

<sup>9</sup> $\overline{h_k}$  can be simply set to 1 for all  $k$ , but we found that better results can be obtained by setting  $\overline{h_k}$  to lower values for earlier layers, and higher values for later layers. This is because later layer tend to output much greater variance in hidden state values - 1 would be considered a very high value in layer 1, but not particularly high in layer 30. Since each transformer layer includes layernorm operations, it's more consistent for the classification target to be set to a value that increases with the layer.

<sup>10</sup>Note that for instruct-formatted examples, we also extend the attention mask to set  $A(i, n) = 0$  when  $n$  corresponds to tokens in user messages. As a result, position loss is only calculated for assistant messages - see section 3 for a rationale.