Probing Prompt Sensitivity in Language Models through Sparse Feature Attribution

Daivik Patel, Shrenik Patel

Abstract

This report explores how semantically identical but syntactically different prompts activate internal representations in large language models. Using Mistral-7B-Instruct-v0.2 and a lightweight Sparse Autoencoder (SAE), we examine gradients with respect to learned features at a middle transformer layer. By comparing top-k SAE feature attributions for 15 paraphrased prompt pairs, we reveal prompt-sensitive behavior and quantify internal representation shifts using Jaccard and cosine similarity. Our results support prior work suggesting that prompt wording steers internal computations, with implications for prompt engineering, model interpretability, and alignment strategies.

1 Introduction

Prompt engineering has emerged as a powerful tool to influence the behavior of large language models (LLMs). However, the mechanistic reasons behind why different prompt phrasings lead to different outcomes remain underexplored. Building upon insights from Goodfire AI's blog on LLaMA-3 steering [1], we investigate how prompt phrasing changes the internal computation of Mistral-7B by analyzing sparse feature attribution through a Sparse Autoencoder (SAE).

2 Methodology

2.1 Model and Layer Selection

We use the open-weight Mistral-7B-Instruct-v0.2 model hosted on HuggingFace. To analyze how representations evolve internally, we extract the hidden state from transformer `layer 16`, which is a middle layer within the 32-layer architecture. This intermediate layer balances surface-level syntax and deeper semantic abstraction, making it ideal for analyzing generalization and internal structure without being overly shallow or task-specific.

2.2 Sparse Autoencoder (SAE)

We construct a lightweight Sparse Autoencoder with a 1024-dimensional bottleneck layer. The SAE is trained to reconstruct the model's residual stream vectors at layer 16, compressing dense activations into interpretable sparse features. This bottleneck creates a compact representation that highlights meaningful variation in the internal computation, and facilitates attribution via gradients (see Appendix D for full setup and training curve).

2.3 Prompt Dataset

We curated a dataset of 15 prompt pairs, where each pair expresses the same semantic intent but uses different wording, tone, or phrasing structure. The prompts span diverse Python programming tasks such as sorting lists, reading files, or building decorators. Each pair is designed to test the model's ability to internally represent the same task under linguistic perturbation (see Appendix A).

2.4 Gradient Attribution and Feature Extraction

For each prompt, we run a forward pass and collect the hidden activations at layer 16. These are passed through the pretrained SAE to yield 1024 sparse features. We then compute the gradient of the predicted logit (for the first generated token) with respect to each of these SAE features. The absolute magnitude of each gradient reflects the importance of that feature in determining the output. We rank the features and retain the top-50 most influential for each prompt (method detailed in Appendix E).

2.5 Similarity Metrics

To quantify representational similarity between prompt variants, we compute two complementary metrics:

- **Jaccard Similarity:** Measures the overlap between the top-50 SAE features of both prompts: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.
- **Cosine Similarity:** Measures the cosine of the angle between the full 1024-dimensional gradient vectors: $\cos(\theta) = \frac{A \cdot B}{|A||B|}$.

These metrics allow us to distinguish between direct feature reuse (Jaccard) and general alignment in representation space (cosine). Results are tabulated in Appendix B.

3 Results

3.1 Summary Statistics

Across the 15 prompt pairs, we observe an **average Jaccard similarity** of **0.3516** and an **average cosine similarity** of **0.8382**. This indicates that while the specific top features often differ between semantically identical prompts, the overall attribution pattern remains quite aligned.

The **minimum Jaccard similarity** (0.0753) occurred for the list comprehension prompts, where one prompt asked for an explanation and the other requested examples—indicating a shift in representational intent. The **maximum Jaccard similarity** (0.7241) occurred for the palindrome checking pair, which used very similar phrasing and triggered nearly identical internal representations.

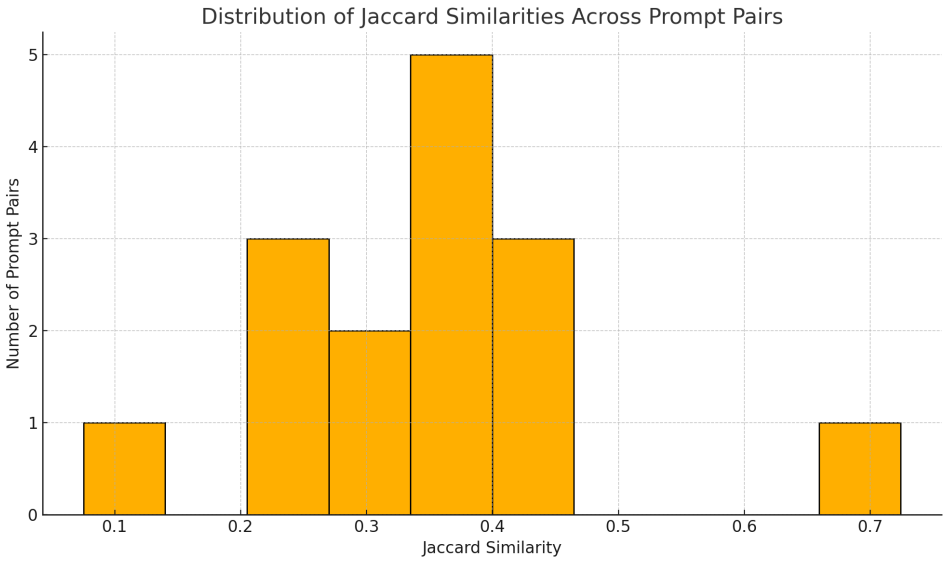

Figure 1: Histogram of Jaccard similarities across 15 prompt pairs.

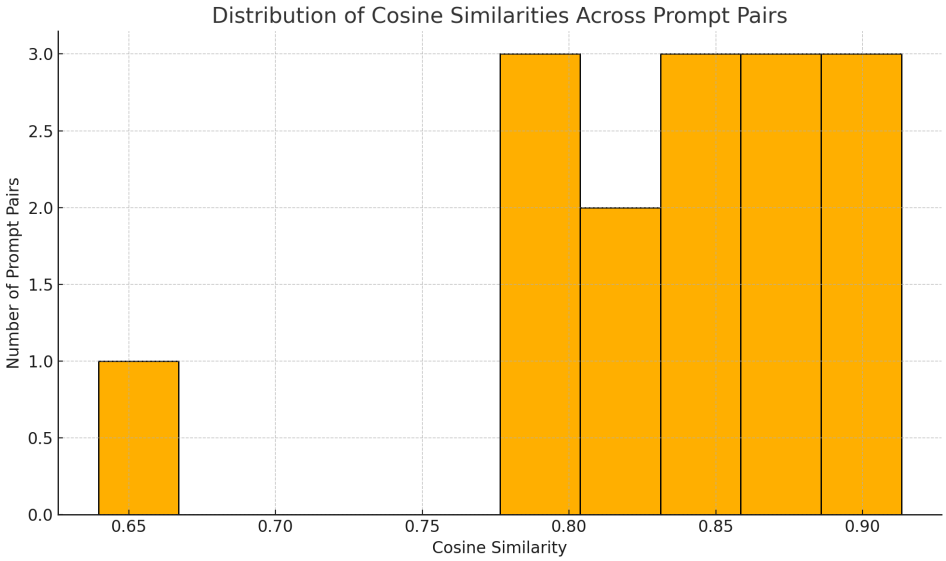

Figure 2: Histogram of Cosine similarities across 15 prompt pairs.

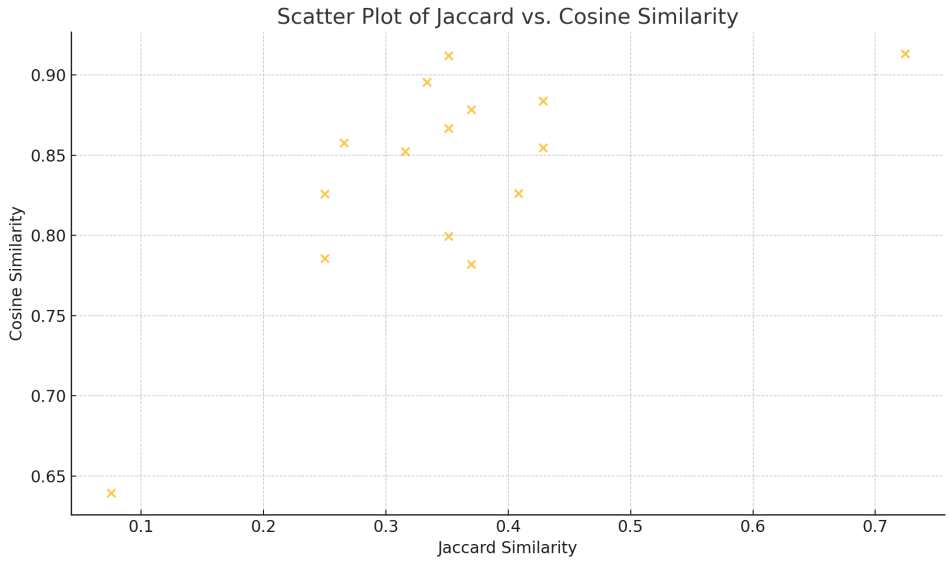

Figure 3: Scatter plot of Jaccard vs. Cosine similarities across prompt pairs.

4 Discussion

4.1 Interpreting Jaccard and Cosine Patterns

Our analysis highlights a key insight: prompt rewrites often yield different high-magnitude SAE features (low Jaccard), yet preserve similar overall directional attribution (high cosine). This suggests a robustness in the model's ability to generalize representations, while also revealing that even small linguistic shifts can re-route internal computation paths.

4.2 Implications for Prompt Engineering

The variation in Jaccard scores confirms that different prompt formulations activate different internal features, even when outputs remain the same. Prompt engineering is thus more than a surface-level heuristic; it is a precise tool for steering model internals. For safety-critical or interpretability-dependent applications, this distinction is crucial.

4.3 Broader Relevance

The methodology introduced here can be extended to evaluate alignment strategies, test refusal tuning, or probe sensitivity to adversarial prompts. It could also help identify representational sparsity patterns and uncover latent semantic clusters in sparse feature space.

5 Conclusion

This study presents a quantitative framework for analyzing prompt sensitivity in LLMs using sparse feature attribution. By comparing gradients over SAE features, we empirically validate that prompt wording significantly affects internal representations. Our contribution includes a reproducible codebase, a curated prompt dataset, and analysis tools for probing model internals. These findings lay groundwork for future work in steerability, transparency, and alignment robustness.

References

[1] Goodfire AI. *Understanding and Steering LLaMA 3*. 2024.
https://www.goodfire.ai/papers/understanding-and-steering-llama-3

A Prompt Pairs

- Write me a Python function that reverses a string.
- Can you show me Python code to reverse a string?
- How do I sort a list in Python?
- Write a Python function to sort a list of numbers.
- Explain how to implement binary search in Python.
- Show me a Python implementation of binary search algorithm.
- How can I read a file in Python?
- Write code to open and read a file in Python.
- Create a Python class representing a simple calculator.
- Implement a calculator class in Python with basic arithmetic operations.
- How do I remove duplicates from a list in Python?
- Write a function to eliminate duplicate elements from a Python list.
- Write a Python script to count word frequency in a text.
- Show me how to count the occurrence of each word in a text using Python.
- Implement a function to check if a string is a palindrome.
- Write Python code to determine whether a string reads the same backward as forward.
- How do I create a web server in Python?
- Write a simple HTTP server using Python.
- Explain how to use list comprehensions in Python.
- Show examples of Python list comprehensions for various operations.
- How can I connect to a SQL database from Python?
- Write Python code to establish a connection with a SQL database.
- Write a Python function to find the factorial of a number.
- Implement a factorial calculator in Python.
- How do I create a RESTful API with Flask?
- Write code for a simple Flask REST API.
- Implement a Python decorator for timing function execution.
- Create a decorator in Python that measures how long a function takes to run.
- How do I parse JSON data in Python?
- Write code to read and process JSON in Python.

B Per-Pair Similarity Metrics

#	Prompt Task	Jaccard	Cosine
1	Reverse String	0.3333	0.8955
2	Sort List	0.2658	0.8576
3	Binary Search	0.4286	0.8546
4	Read File	0.3514	0.9122
5	Calculator Class	0.3699	0.7821
6	Remove Duplicates	0.3699	0.8784
7	Word Frequency	0.3514	0.8668
8	Palindrome Check	0.7241	0.9132
9	Web Server	0.3514	0.7997
10	List Comprehension	0.0753	0.6396
11	SQL Connection	0.2500	0.7857
12	Factorial	0.4286	0.8838
13	Flask API	0.3158	0.8523
14	Decorator Timer	0.2500	0.8258
15	JSON Parsing	0.4085	0.8263

Table 1: Jaccard and Cosine similarities for all 15 prompt pairs.

C Top-K Feature Attribution Example

Below is an example of the top-50 SAE feature indices for the palindrome check prompt pair:

Prompt 1: `Implement a function to check if a string is a palindrome.`

Prompt 2: `Write Python code to determine whether a string reads the same backward as forward.`

Top-50 Intersection (features appearing in both):

```
[256, 515, 261, 270, 895, 795, 797, 287, 672, 298, 427, 812, 688, 562, 693, 183,
314, 960, 705, 68, 458, 590, 342, 858, 350, 95, 223, 608, 607, 991, 356, 997,
869, 736, 491, 877, 628, 119, 376, 762, 507, 767]
```

D SAE Architecture and Training Setup

- **Input:** Residual stream from transformer layer 16 (Mistral-7B).
- **Bottleneck:** 1024-dimensional sparse representation.
- **Decoder:** Symmetric fully connected layer.
- **Activation:** ReLU.
- **Loss Function:** Mean Squared Error (MSE) + ℓ_1 sparsity penalty.
- **Optimizer:** Adam.
- **Epochs:** 35.
- **Final Loss (Epoch 3):** 1.957.
- **Training Time:** Approx. 4.5 hours.

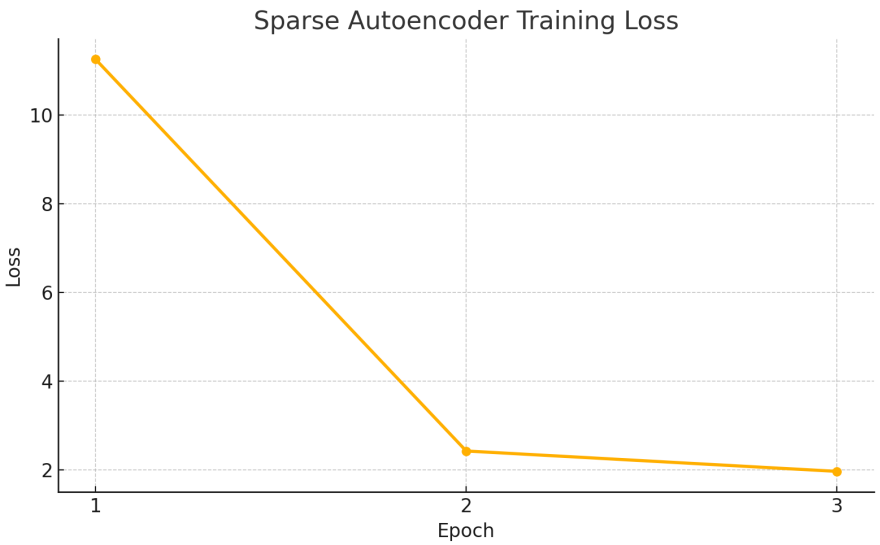

Figure 4: Training loss over epochs for the Sparse Autoencoder. Loss steadily decreases, indicating successful compression of hidden state features.

E Gradient Attribution Method

For each prompt, we compute:

- The logit of the **first predicted token**.
- The gradient of that logit with respect to each SAE feature.
- Feature importance is calculated using the **absolute gradient magnitude**.
- The **top-50 features** with the highest values are selected.

This process quantifies how much each sparse feature contributes to the first token's generation.

F Environment and Runtime

- **Model:** Mistral-7B-Instruct-v0.2
- **SAE Training Time:** 4.5 hours
- **Inference + Attribution Time:** 30 minutes (15 prompt pairs)
- **Hardware:** Apple M4 Mac Pro chip
- **Libraries:** PyTorch 2.1, HuggingFace Transformers, NumPy, Matplotlib
- **Platform:** Ubuntu 20.04, Python 3.10