Understanding LLM Behavior on HPC Data via Mechanistic Interpretability

Md Mahbubur Rahman [1], Arjun Guha [2], Harshitha Menon [3]
mdrahman@iastate.edu, a.guha@northeastern.edu, harshitha@llnl.gov
[1] Iowa State University, [2] Northeastern University, [3] Lawrence Livermore National Laboratory

Large Language Models (LLMs) are increasingly used in HPC for tasks like code generation and analysis, but their internal reasoning remains opaque. To address this, we study three tasks—OpenMP code completion, data race detection and OMP code generation—using mechanistic interpretability. Sparse autoencoder ablations reveal causal features, function vector injection improves zero-shot predictions and direction vector shifts the model's output toward a desired behavior or style, even without explicitly stating it in the prompt. These methods expose and influence LLM behavior in HPC contexts.

INTRODUCTION

Large Language Models (LLMs) are increasingly used in High-Performance Computing (HPC) for coding tasks. Despite their strong performance, it is often unclear how LLMs make decisions, which poses risks in critical domains. To address this, we turn to mechanistic interpretability: an approach focused on reverse-engineering the internal components of neural networks to understand why they behave the way they do.

In this work, we use three interpretability methods to examine the models during HPC tasks.

- Feature-Level Analysis with Sparse Autoencoders – identifies which internal components help the model complete code correctly.
- Representation Steering with Function Vectors – guides model predictions by injecting representations from few-shot examples.
- Direction Vector – shift the model's output to OMP direction during HPC code generations.

BACKGROUND

Sparse Autoencoder Feature Analysis

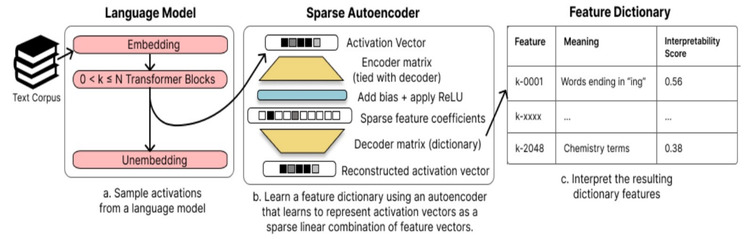

Figure 1: Internal model activations (e.g., from the residual stream or attention layers) are used to train a sparse autoencoder, which learns a set of sparse, disentangled features. These features can then be interpreted using techniques like auto-interpretability scores to reveal their functional roles. (Cunningham et al., 2023)

Function Vector Steering

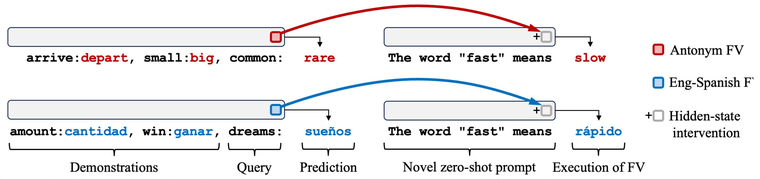

Figure 2: Average activations are computed from a set of few-shot prompts; when added to a zero-shot context, they influence the model to produce behavior aligned with the few-shot examples. (Todd et al., 2024)

Direction Vector

- *At a specific layer, compute the direction by subtracting the mean activation of class 2 from that of class 1 (averaged over the last few tokens and across n prompts).*
- *Adding this vector shifts the model's outputs toward **class 1**; while adding the opposite vector shifts it toward **class 2**. (Arditi et al.)*

METHODS

Sparse Autoencoder Feature Analysis

- Task type: Fill-in-the-Middle (FIM)
- *Prompt:*

```
#pragma omp parallel for reduction(+: <FILL_ME> )      // predict: sum
for (i = i2; i < (((i2 + 256) < (len)) ? (i2 + 256) : (len)); i++)
    sum += a[i] * b[i];
```

- From the SAE, identify features that are frequently activated with high scores
- Examine their causal role by selectively turning them off
- A significant drop in accuracy indicates these features causally contribute to the model's reasoning.

Function Vector Steering

- Task type: Classification
- *Prompt:*

N Shot Prompting

```
You are an expert OpenMP code reviewer. For any code snippet
you receive, decide exactly one thing:
• Does it require additional synchronization to be data-race free?
• Please response with yes or no.
Code snippet:
```
Shot 1
```
Output: YES/NO
Code snippet:
```
Shot n
```
Output: YES/NO
Code snippet:
```
#pragma omp parallel for
    for (i = 0; i < len; i++) {
        tmp = a[i] + i;
        a[i] = tmp;
    }
```
Output:
```

Zero Shot Prompting

```
You are an expert OpenMP code reviewer. For any code
snippet you receive, decide exactly one thing:
• Does it require additional synchronization to be data-race
free?
• Please response with yes or no.
Code snippet:
```
#pragma omp parallel for
    for (i = 0; i < len; i++) {
        tmp = a[i] + i;
        a[i] = tmp;
    }
```
Output:
```

Direction Vector

- Task type: OMP Code Generation
- *OMP Prompt:*
  Return the distance between the closest two points in the vector points.
  Use OpenMP to compute in parallel.
  Source code: ```
- *Serial Prompt:*
  Return the distance between the closest two points in the vector points.
  Source code: ```
- Compute the OMP direction vector for each layer individually.
- Select the best direction vector by evaluating on the validation set.
- Similarly, we evaluate a language direction contrasting Python and C++ using the same setup.

EXPERIMENTAL DETAILS & RESULTS

Sparse Autoencoder Feature Analysis

- Model: Llama-3 8B
- SAE: Layer 27, sparse factor 32
- Dataset: Code snippets are collected from DataRaceBench, (Liao et al., 2017), total data: 126

Table 1. Result of the sparse autoencoder's feature analysis.

Methods	#of correct predictions
Base Model	112
Base Model + SAE	107
Base Model + Individual feature turn off	102
Base Model + Group of features turn off	97
Base Model + Group of random features turn off	106

Function Vector Steering

- Model: Llama-3 8B
- Dataset: Code snippets are collected from DataRaceBench, (Liao et al., 2017). Each class has 131 data.
- Number of shots: 8

Table 2. Result of the function vector steering.

Methods	#of correct positive predictions	#of correct negative predictions
Zero Shot	100	49
8 Shots	109	103
Zero Shot + FV	97	75
Zero Shot + Rand. Noise	106	33

Direction Vector

- Model: CodeGemma 7B
- Dataset:
  - OpenMP vs Serial: ParEval, validation data - 30, test data - 30
  - Python vs CPP: MultiPL-E, validation data - 81, test data - 81
- Results:
  - OMP Direction: 83% aligned with OpenMP code generation
  - Serial Direction: 100% aligned with serial code generation
  - Python Direction: 97.53% codes are generated in python
  - C++ Direction: 92.53% codes are generated in C++

CONCLUSIONS

- Disabling key sparse autoencoder features leads to reduced code completion accuracy, indicating their causal role.
- Direction vectors effectively shift model outputs toward desired code styles (e.g., OpenMP or serial), demonstrating controllable behavior.

[1] Cunningham, Hoagy, et al. "Sparse autoencoders find highly interpretable features in language models." *arXiv preprint arXiv:2309.08600* (2023).
[2] Todd, Eric, et al. "Function vectors in large language models." *arXiv preprint arXiv:2310.15213* (2023).
[3] Arditi, Andy et al. "Refusal in Language Models Is Mediated by a Single Direction", https://arxiv.org/abs/2406.11717, (2024)
[4] Liao, Chunhua, et al. "DataRaceBench: a benchmark suite for systematic evaluation of data race detection tools." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2017.

Lawrence Livermore National Laboratory

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, through solicitation DE-FOA-0003264, "Advancements in Artificial Intelligence for Science," under Award Number DE-SC0025598. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL) under Contract DE-AC52-07NA27344 (LLNL-POST-2011890).

National Nuclear Security Administration