

◆ Context

AI-assisted coding has achieved **widespread adoption**:

- 30% of AI-suggested code accepted into production¹
- Projected \$1.5T GDP boost by 2030¹

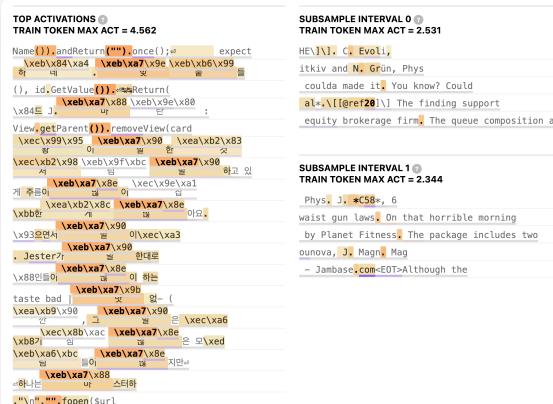
◆ The Problem

LLMs' internal mechanisms for code correctness remain poorly understood.

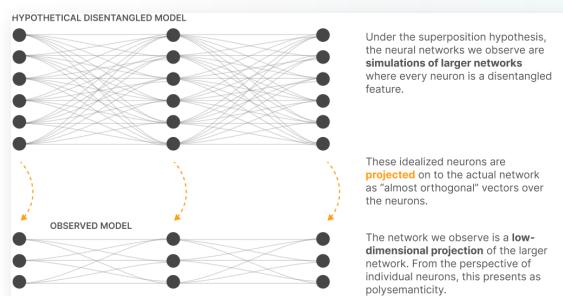
- 44% of LLM bugs identical to historical training errors²
- Only 12.27% accuracy in bug-prone contexts²
- Critical for **high-stakes systems** (healthcare, banking, military) demanding transparency

◆ The Challenge

Understanding LLMs requires analyzing individual neurons—but neurons are **polysemantic**, responding to multiple unrelated concepts like academic citations, English dialogue, HTTP requests, and Korean text.

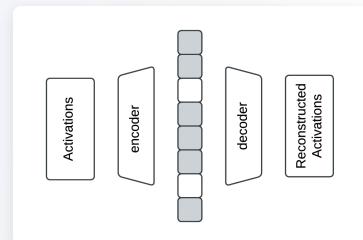


A hypothesized cause is **superposition**: networks encode more features than available dimensions. The observed model is a low-dimensional projection of a larger, idealized network where features would be disentangled.



◆ Our Approach

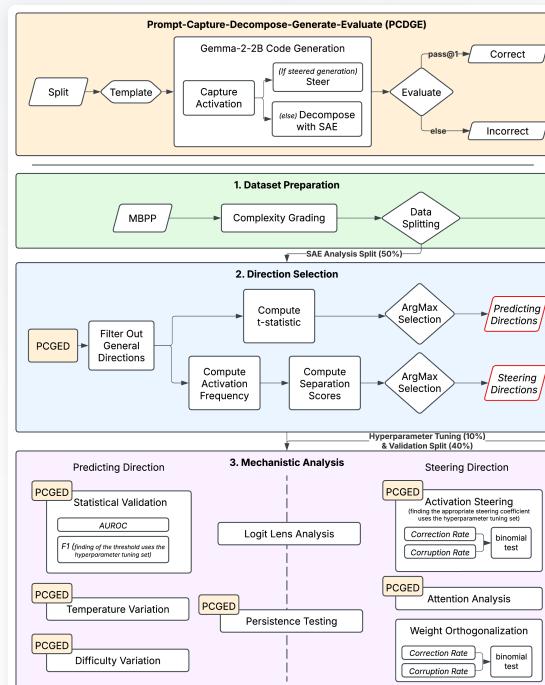
Sparse Autoencoders (SAEs) address superposition by expanding activations into a higher-dimensional sparse space, decomposing entangled representations into interpretable directions.



SAE: Activations → sparse latent space → reconstructed activations

◆ Methodology Pipeline

Using 1,000 Python problems from MBPP, we capture residual stream activations at the final prompt token across all layers, then identify two predicting directions (correct/incorrect, via t-statistics) and two steering directions (correct/incorrect, via separation scores).



◆ Key Discovery

Code correctness directions **EXIST** in LLM representations, revealing an **asymmetry**.

Identified Directions

Direction	Layer	Metric	Result
Incorrect Predicting	L19	t=5.68	F1=0.821 ✓
Correct Predicting	L16	t=5.09	F1=0.504 ✗
Correct Steering	L16	sep=0.22	4.04% (p<0.001) ✓
Incorrect Steering	L25	sep=0.20	2.2% < control ✗

*Note: Correct steering also corrupts 14.66% of initially correct code, suggesting selective application.

Asymmetric Finding

Found:	incorrect-predicting & correct-steering
Not Found:	correct-predicting & incorrect-steering

The asymmetry works in our favor: we can detect errors AND steer toward correctness

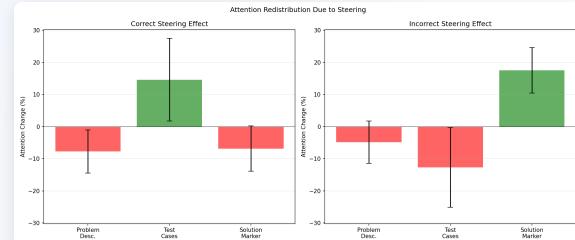
◆ Mechanistic Evidence

Attention Analysis

Our prompts contain three components:

```
Problem   Write a function to find the minimum cost path to reach (m, n) from (0, 0) from the given cost matrix cost[][] and a position (m, n) in cost[][].
Description
Test Cases "assert min_cost([[1, 2, 3], [4, 8, 2], [1, 5, 3]], 2, 2) == 8",
             "assert min_cost([[2, 3, 4], [5, 9, 3], [2, 6, 4]], 2, 2) == 12",
             "assert min_cost([[3, 4, 5], [6, 10, 4], [3, 7, 5]], 2, 2) == 16"
Code Initiator # Solution:
```

Where does the model focus attention when steering activates?



Correct-steering redirects attention to **test cases** (+15%), while **incorrect-steering** shifts away (-13%). *Implication: Prompting should prioritize test examples.*

Weight Orthogonalization

Is the direction merely correlated, or **necessary**? We surgically remove each from model weights.



*Correct-steering is **necessary** for generation; incorrect-steering removal doesn't fix errors (asymmetry confirmed).*

Persistence Across Fine-tuning

Do these directions persist from base to instruction-tuned models?



Both directions persist through fine-tuning, confirming these are fundamental mechanisms.

◆ Significance

First application of Sparse Autoencoders to study code correctness mechanisms in LLMs.

Practical Applications

1. Prompting strategies: Prioritize test examples over problem descriptions

2. Error alarms: Predictor directions flag code for review

3. Selective steering: Intervene only when errors anticipated

Safety Implications: Contributes to safer AI deployment in healthcare, finance, and critical infrastructure.

References

- Dohmke et al. (2023). Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle.
- Guo et al. (2025). An Empirical Study on LLM-Generated Bug Analysis.
- Bricken et al. (2023). Towards Monosematicity: Decomposing Language Models With Dictionary Learning.
- Templeton et al. (2024). Scaling Monosematicity: Extracting Interpretable Features from Claude 3 Sonnet.
- Lieberum et al. (2024). Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2.
- Ferrando et al. (2024). A Primer on the Inner Workings of Transformer-based Language Models.

Mechanistic Interpretability

of Code Correctness in LLMs

via Sparse Autoencoders



Kriz Tahimic

Adviser: Dr. Charibeth K. Cheng

College of Computer Studies
De La Salle University

Academic Year 2025–2026

Contact

PROONENT
Kriz Tahimic
kriz_tahimic@dlsu.edu.ph

ADVISER
Dr. Charibeth K. Cheng
charibeth.cheng@dlsu.edu.ph

College of Computer Studies
De La Salle University, Manila
BS Computer Science, Software Technology