## Mechanistic Evidence

Multiple analyses validate our findings:

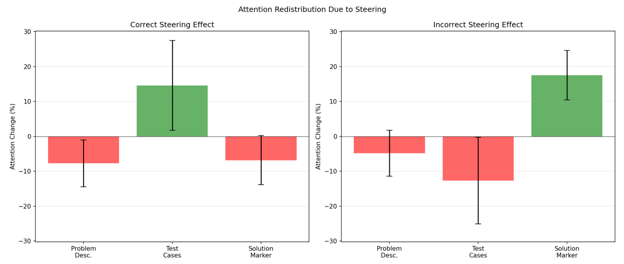
### Prediction Analysis

- Incorrect-predicting: **F1 = 0.821**
- Correct-predicting: F1 = 0.504 (weak)

### Steering Interventions

- Correct-steering fixes 4.04% of errors
- Trade-off: affects 14.66% correct code

### Attention Analysis



Test cases matter MORE than problem descriptions

### Necessity (Orthogonalization)

- Removing correct directions: 83.62% code corruption
- Control removal: only 18.97% corruption

### Persistence Across Fine-tuning

- Base $\rightarrow$ Instruction-tuned: F1 0.821 $\rightarrow$ 0.772
- Mechanisms learned in pre-training persist

## 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 developer review
3. **Selective steering**: Intervene only when errors anticipated

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

## Key References

- Bricken et al. (2023) — Towards Monosemanticity
- Templeton et al. (2024) — Scaling Monosemanticity
- Lieberum et al. (2024) — GemmaScope
- Ferrando et al. (2024) — Entity Recognition via SAEs

## Contact

**Proponent**:
Kriz Royce 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, Philippines

BS Computer Science — Software Technology

# Mechanistic Interpretability of Code Correctness in LLMs

via Sparse Autoencoders

**Kriz Royce Tahimic**

Adviser: Dr. Charibeth K. Cheng

College of Computer Studies
**De La Salle University**

Academic Year 2024–2025

## Context

AI-assisted coding has reached critical mass:

- **30%** of GitHub Copilot suggestions enter production code
- Projected **$1.5 trillion** GDP impact by 2030
- LLMs increasingly deployed in critical systems

## The Problem

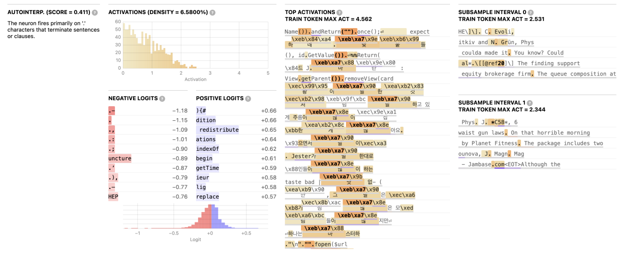
> We lack **mechanistic understanding** of WHEN and WHY models produce correct code.

- Models fail in bug-prone contexts (12.27% accuracy)
- 44% of LLM bugs mirror historical training bugs
- Critical for: healthcare, banking, military

## The Challenge

Neural networks are hard to interpret:

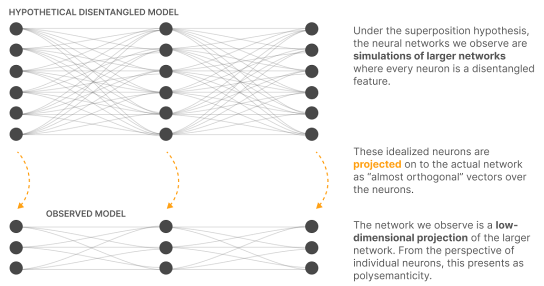
- **Polysemantic neurons**: One neuron responds to multiple unrelated concepts


AUTOINTERP. (SCORE = 0.411)
The neuron fires primarily on ',' characters that terminate sentences or clauses.
ACTIVATIONS (DENSITY = 6.5800%)
NEGATIVE LOGITS
POSITIVE LOGITS
TOP ACTIVATIONS
TRAIN TOKEN MAX ACT = 4.562
SUBSAMPLE INTERVAL 0
TRAIN TOKEN MAX ACT = 2.531
SUBSAMPLE INTERVAL 1
TRAIN TOKEN MAX ACT = 2.344

A single neuron activates for unrelated concepts (cat, car, citation)

## Why Interpretation is Hard

**Superposition**: Networks compress more features than dimensions, entangling representations.


HYPOTHETICAL DISENTANGLED MODEL
Under the superposition hypothesis, the neural networks we observe are simulations of larger networks where every neuron is a disentangled feature.
These idealized neurons are projected on to the actual network as "almost orthogonal" vectors over the neurons.
OBSERVED MODEL
The network we observe is a low-dimensional projection of the larger network. From the perspective of individual neurons, this presents as polysemanticity.

Features compressed into overlapping directions

## Our Approach

**Sparse Autoencoders (SAEs)** decompose entangled representations into interpretable directions.

> **Key Idea**: If code correctness is represented as a *direction* in the model's latent space, we can:
> 1. **Detect** it (prediction)
> 2. **Manipulate** it (steering)
> 3. **Validate** it (causal analysis)

**Technical Setup**:

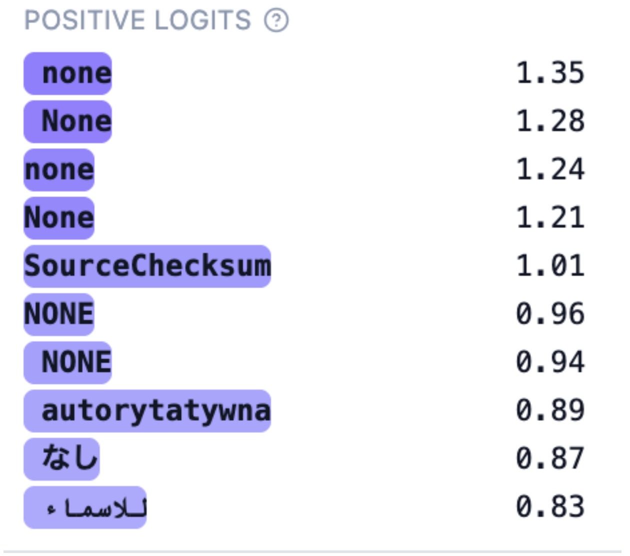
- **Model**: Gemma-2 2B (base & instruction-tuned)
- **SAE**: GemmaScope pre-trained (16K features)
- **Dataset**: MBPP (1,000 Python problems)
- **Analysis**: Residual stream at layer 20

## Key Discovery

**Code correctness directions EXIST in LLM representations—and they are actionable.**

### 1. Predict Errors Before Generation

The **incorrect-predicting direction** detects errors with high accuracy:



POSITIVE LOGITS

| | |
|---|---|
| none | 1.35 |
| None | 1.28 |
| none | 1.24 |
| None | 1.21 |
| SourceChecksum | 1.01 |
| NONE | 0.96 |
| NONE | 0.94 |
| autorytatywna | 0.89 |
| なし | 0.87 |
| ـلاسلا، | 0.83 |

F1: 0.821 for error detection—can serve as "error alarm"

### 2. Steer Toward Correctness

The **correct-steering direction** can fix errors (4.04% of incorrect code fixed).

### 3. Asymmetric Finding

- Found: **incorrect-predicting** + **correct-steering**
- Did NOT find: correct-predicting or incorrect-steering
- *Models detect "wrongness" differently than they encode "correctness"*