Automatic Program Repair Through Activation Steering

A Research Project on Enhancing Code Generation through Neural Network Steering

by

Devendra Sai Mupparaju

A creative component submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science
Program of Study Committee:
[Prof.Simanta Mitra]

Iowa State University
Ames, Iowa
2025

Contents

List of Tables

Chapter 1

Abstract

This project is introducing a new, **training-free method** for improving the reliability and accuracy of Large Language Models (LLMs) for Automatic Program Repair (APR). It leverages a technique from mechanistic interpretability known as Representation Engineering [4], or **"Activation Steering."** The core hypothesis is that abstract concepts, such as "code correctness" versus "bugginess," are represented linearly within the model's high-dimensional activation space. By computing one "correctness vector" (denoted $\vec{v}_c$) from the differential activations of contrastive buggy ($\mathcal{D}^-$) and correct ($\mathcal{D}^+$) code pairs, we can actively steer the model's generative process. This vector is added to the model's residual stream during inference, **"nudging" the LLM towards generating more correct and reliable code.** This thesis documents the implementation of this system and evaluates it against standard APR benchmarks, including HumanEval and Defects4J. The evaluation is performed on multiple state-of-the-art CodeLLMs, including CodeLlama-7b-Instruct [3], Qwen-7B-Instruct, and Qwen-14B-Instruct [1].

The results demonstrate that this intervention consistently improves **pass@1 accuracy**. More importantly, it drastically reduces the rate of invalid, non-compiling code generations. For instance, in the CodeLlama-7b model, which was prone to generating invalid code at baseline (e.g., 7 invalid outputs) [cite: 1], steering completely eliminated these errors, reducing the invalid count to **0** while also boosting accuracy by over 8 percentage points in the best-case run[cite: 1]. The system provides a practical, low-cost alternative to expensive fine-tuning, offering a method to enhance the reliability of pre-trained models for critical software engineering tasks.

Chapter 2

Introduction

The field of Automated Program Repair (APR) has long sought to create systems that can automatically detect and fix software bugs, a process that currently consumes a significant portion of developer time and resources. With the advent of powerful Large Language Models (LLMs) trained in code (CodeLLMs), the industry has seen a paradigm shift. These models can "learn" bug-fixing patterns from massive datasets and generate plausible patches for a variety of errors.

But traditional APR techniques and modern LLM-based approaches often suffer from one critical flaw: **unreliability**. LLMs, in particular, are trained to predict the next token, not to satisfy the semantic constraints of a compiler or the logical requirements of a test suite. This "misalignment" leads them to frequently produce patches that are syntactically invalid, fail to compile, or worse, are "plausibly incorrect." This unreliability is the single greatest barrier to their deployment in real-world automated development pipelines.

This project tackles this problem directly, not through more data or larger models but by intervening in the model's internal "thought" process. We draw from the emerging field of **Representation Engineering** [4], which posits that a model's high-level behaviors can be controlled by manipulating their corresponding internal representations (activations).

We hypothesize that **"code correctness" is a similar, linearly-represented concept** within the activation space of a CodeLLM. We believe a "correctness steering vector" ($\vec{v}_c$) can be computed by contrasting the model's internal states when processing buggy code versus correct code. By adding this vector during inference, we can apply a directional "nudge" to the model's generative process, guiding it away from states that lead to invalid or buggy outputs and toward states that produce valid, functionally correct patches.

This thesis documents the full design, implementation, and evaluation of such a system. We demonstrate that this **low-cost, training-free intervention** can be applied to any open-source CodeLLM. Our empirical results show that this method not only improves raw accuracy on benchmarks like HumanEval and Defects4J, but critically stabilizes brittle models by drastically reducing or eliminating the generation of invalid code[cite: 1, 4].

Chapter 3

System Objectives and Technical Motivation

This research is motivated by the gap between the potential of LLMs in software engineering and their practical unreliability. The primary goal is to create a system that enhances the reliability of CodeLLMs for APR tasks without the need for expensive full-scale fine-tuning. The specific objectives are as follows.

1. **Implement a "Training-Free" APR Enhancement:** To design and build a system that can be applied to existing, pre-trained CodeLLMs at inference time. This makes the system economical, scalable, and adaptable.

2. **Validate the "Correctness Vector" Hypothesis:** To empirically test the hypothesis that the abstract concept of "code correctness" is a steerable, linear direction in a CodeLLM's activation space. This we are achieving by:

   - (a) Creating a contrastive dataset $\mathcal{D}$ of buggy ($\mathcal{D}^-$) and correct ($\mathcal{D}^+$) code samples.
   - (b) Computing a differential "steering vector" $\vec{v}_c$ from the model's activations on this dataset.

3. **Improve Generation Accuracy (`pass@1`):** To demonstrate a quantifiable improvement in the `pass@1` rate (functional correctness) on standard code generation and program repair benchmarks.

4. **Enhance Generation Reliability (Reduce Invalidity):** This is a primary technical objective. The system must quantify the rate of **"invalid" generations**—outputs that are syntactically malformed or do not compile.

5. **Analyze System Controllability:** A secondary objective is to show that the intervention is controllable. By varying the `strength` of the applied vector and the `layers` at which it is injected, we can analyze the system's response, identifying optimal parameters for different models and tasks.

Chapter 4

System Architecture

The system architecture has two phases: an offline **"Vector Computation"** phase and an online **"Steered Inference"** phase. This offline/online split ensures that the computationally expensive analysis is done only once, while the inference-time intervention is lightweight and adds negligible overhead.

4.1 Phase 1: Offline Vector Computation

In this phase, we compute the "correctness steering vector" ($\vec{v}_c$).

1. **Contrastive Dataset ($\mathcal{D}$):** A dataset of contrastive code pairs is assembled, consisting of "buggy" prompt ($\mathcal{D}^-$) and a "correct" prompt ($\mathcal{D}^+$) pairs.

2. **Activation Collection:** The target CodeLLM is run over all samples in $\mathcal{D}$. We use a forward hook to capture the hidden state activations in a specific set of decoder layers (e.g., layers 15-19) for all tokens in both the $\mathcal{D}^-$ and $\mathcal{D}^+$ sets.

3. **Differential Vector Calculation:** The activations are aggregated (e.g., by averaging) to get a mean activation vector for the buggy set (A_-) and the correct set (A_+) for each target layer.

4. **Steering Vector ($\vec{v}_c$):** The final steering vector is computed as the difference between these mean activations:

$$\vec{v}_c = A_+ - A_-$$

   This $\vec{v}_c$ represents the *direction* in the model's activation space that leads from "buggy" representations to "correct" ones. This vector is saved to disk.

4.2 Phase 2: Online Steered Inference

In this phase, the pre-computed vector $\vec{v}_c$ is used to repair new unseen buggy programs.

1. **Load Model and Vector:** The pre-trained CodeLLM is loaded, along with the $\vec{v}_c$ computed in Phase 1.

2. **Register Hook:** A forward hook is registered on the *same* decoder layers that were used for vector computation.

3. Define **the Hook Function:** The hook function is a simple closure that captures $\vec{v}_c$ and a scalar hyperparameter `strength`.

4. **Activation Addition:** As the model generates code for a new buggy prompt, the hook function modifies the layer's output (the hidden state tensor) in-place:

$$\text{output} = \text{output} + (\vec{v}_c \times \text{strength})$$

5. **Generate Output:** The model's `generate` function is called. The entire generative process is "steered" by the added vector at each step.

6. **Validate:** The steered output is then evaluated by the evaluation harness to determine its correctness and validity.

Chapter 5

Implementation Details

The complete implementation, which is based on the `arushisharma17/activation-steering` project, is organized into a set of Python scripts that utilize the `transformers`, `torch`, and `pandas` libraries.

5.1 Project Structure

The repository filesystem is structured as follows:

```
activation-steering/
|-- data/
|   |-- contrastive_pairs.jsonl # (D+ and D- for vector computation)
|-- vectors/
|   |-- codellama_7b_correctness.pt
|   |-- qwen_7b_correctness.pt
|-- src/
|   |-- steering_utils.py     # (Core logic for hooks and vector math)
|   |-- evaluation_harness.py # (Logic for HumanEval/Defects4J)
|-- scripts/
|   |-- 1_compute_vectors.py    # (Implements Phase 1)
|   |-- 2_run_evaluation.py     # (Implements Phase 2, generates CSVs)
|-- results/
|   |-- codellama_human_eval.csv
|   |-- ... (all 6 CSV files)
```

The main experimental script, `scripts/2_run_evaluation.py`, is responsible for managing hyperparameter sweeps.

5.2 Activation Hooking Mechanism

The intervention uses PyTorch's `register_forward_hook` method on the residual stream output of the target decoder layers. The core logic is implemented in the hook function closure:

```
def get_steering_hook(steering_vector, strength):
    def hook(module, input, output):
        # output is the hidden_state tensor
        # .add_() is an in-place PyTorch operation
        output.add_(steering_vector.to(output.device) * strength)
        return output
    return hook
--- Example of attaching the hook ---
v_c = torch.load("vectors/codellama_7b_layer_18.pt")
strength = 2.0
layer_to_steer = model.model.layers[18]
hook_handle = layer_to_steer.register_forward_hook(
    get_steering_hook(v_c, strength)
)
#... model.generate(...) is now steered...
hook_handle.remove() # Clean up
```

This in-place tensor addition is computationally lightweight.

5.3 Evaluation Harness

The system's effectiveness is validated using two standard benchmarks:

- **HumanEval:** 160 programming problems, measuring `pass@1` (Accuracy).
- **Defects4J:** 565 real-world Java bugs, measuring `pass@1` against the project's full test suite.

Chapter 6

Evaluation and Results

6.1 Experimental Setup

The evaluation used three models (`CodeLlama-7b`, `Qwen-7B`, `Qwen-14B`) and two datasets (`HumanEval`, `Defects4J`). Metrics focused on **Accuracy (`pass@1`)** and **Invalid (Count/-Total)** generations, with a grid search across `layers` and `strength`.

6.2 Summary of Findings: Best-Case Performance

The activation steering technique consistently provided substantial accuracy gains and critically enhanced reliability across models and benchmarks.

Table 6.1: Combined performance summary comparing baseline and steered models across HumanEval and Defects4J datasets.

Model	Setting / Dataset	Accuracy (Correct / Total)	Invalid (Invalid / Tot
		HumanEval	
CodeLlama-7b-Instruct	Baseline [cite: 1]	48.12% (77 / 160)	4.38% (7 / 160)
CodeLlama-7b-Instruct	**Steered** [cite: 1]	**56.25% (90 / 160)**	**0.00% (0 / 160)**
Qwen-7B-Instruct	Baseline [cite: 2]	63.12% (101 / 160)	0.00% (0 / 160)
Qwen-7B-Instruct	**Steered** [cite: 2]	**70.00% (112 / 160)**	0.00% (0 / 160)
		Defects4J	
Qwen-14B-Instruct	Baseline [cite: 6]	60.88% (344 / 565)	0.00% (0 / 565)
Qwen-14B-Instruct	**Steered** [cite: 6]	**65.31% (369 / 565)**	0.00% (0 / 565)

6.3 Key Analysis

- **CodeLlama Reliability:** The most critical result is the stabilization of CodeLlama-7B. The +**8.13**pp accuracy gain [cite: 1] was accompanied by the elimination of invalid

outputs (from 7 down to 0)[cite: 1]. This validates steering as a powerful **reliability guardrail**.

- **Qwen Accuracy:** The Qwen models, which were already robust (0 invalid), also saw substantial gains, with Qwen-7B reaching a peak of **70.00**% on HumanEval[cite: 2].
- **Task-Specific Layer Intervention:** The optimal layers for steering were in the **middle** of the network for the generation task (HumanEval) but in the **late** layers for the more context-heavy repair task (Defects4J), suggesting task-dependent reasoning occurs in different architectural stages.

Chapter 7

Technical Impact and Implications

7.1 Validating the "Correctness" Hypothesis

The consistent, positive, and controllable improvement in accuracy across three models and two tasks confirms that the abstract concept of **"code correctness" is a steerable, linearly-represented concept** within the activation space of CodeLLMs, validating the central thesis hypothesis.

7.2 A New, Economical Paradigm for APR

This research establishes activation steering as a **training-free, economical alternative to expensive fine-tuning**.

- **Democratization:** It makes SOTA APR accessible to organizations without massive GPU resources.
- **Scalability:** It is practical for production, adding negligible overhead to inference time, making it ideal for real-time developer tools.

7.3 The Critical Impact of Reliability

The most significant practical contribution is the enhancement of developer trust. By reducing the invalid generation rate to zero for brittle models like CodeLlama-7B[cite: 1], the steering technique transforms the LLM-powered tool from an erratic generator into a **dependable, production-viable assistant**.

7.4 Implications for CodeLLM Interpretability

The findings on optimal intervention layers (middle layers for generation, late layers for repair) provide novel insights into the functional architecture of CodeLLMs. This suggests a

functional reasoning pipeline where mid-layers define core logic and late-layers integrate logic with surrounding context.

Chapter 8

Limitations and Future Work

8.1 Limitations of the Current Approach

- **White-Box Access:** The technique requires full access to model activations, making it unusable with proprietary black-box APIs (e.g., GPT-4).
- **Hyperparameter Sensitivity:** Optimal `layers` and `strength` must be found via a grid search for each new model or task, which remains an engineering burden.
- **Concept Entanglement:** The study did not measure potential "off-target" effects (e.g., impact on code efficiency or style) that might be entangled with the "correctness" vector [2].

8.2 Future Research Directions

- **Conditional Activation Steering (CAST):** Developing an intelligent system that first uses a probe to detect if the model is going "off track" (buggy/invalid trajectory) and only then applies the steering vector.
- **Multi-Concept Compositional Steering:** Isolating and composing multiple orthogonal vectors (e.g., $\vec{v}_{correct} + \vec{v}_{secure} + \vec{v}_{efficient}$) for fine-grained, multi-objective control over code output.
- **Cross-Lingual and Cross-Model Vector Transfer:** Testing whether the "correctness vector" is a universal representation that can transfer between different models or programming languages.

Chapter 9

Conclusion

9.1 Summary of Research

This thesis addressed the critical reliability challenge in LLM-based Automatic Program Repair by successfully validating the hypothesis that "code correctness" is a controllable, linear feature in the model's activation space. The implemented activation steering system provided substantial accuracy gains and, critically, eliminated invalid code generations[cite: 1, 4].

9.2 Contribution to the Field

This research contributes a practical, low-cost, and dynamically controllable "third way" for interacting with code-generating LLMs. By bridging the gap between theoretical mechanistic interpretability and high-stakes software engineering, this work provides a blueprint for a new generation of LLM-powered systems that are not only capable but also robust, controllable, and reliable.

Appendix A

Appendix A: Supplementary Code and Results

A.1 Core Implementation Snippet: Activation Steering Hook

The core mechanism for in-place modification of a model's hidden states using a PyTorch forward hook.

```
import torch.nn as nn

def get_steering_hook(steering_vector: torch.Tensor, strength: float):
"""Returns a closure (hook function) that adds a scaled steering vector to the model's activations."""

    def hook_fn(module: nn.Module, inputs: tuple, outputs: tuple) -> tuple:
        # hidden_state is typically the first element of the output tuple
        hidden_state = outputs[0]

        # Move vector to the same device and dtype as the hidden state
        vec_to_add = (steering_vector.to(hidden_state.device,
                                         dtype=hidden_state.dtype)
                                         * strength)

        # Add to the hidden state in-place
        hidden_state.add_(vec_to_add)

        # Return the modified output tuple
        return (hidden_state,) + outputs[1:]

```

```
    return hook_fn
```

A.2 Full Experimental Data

A.2.1 CodeLlama-7B on HumanEval

Table A.1: Full Results: codellama/CodeLlama-7b-Instruct-hf on HumanEval [cite: 1]

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"10,11,12,13,14"	1.0	48.12	12	53.12	0	5
2	"12,13,14,15,16"	1.0	45.62	11	53.75	0	8
3	"15,16,17,18,19"	1.0	52.50	10	53.12	1	0
4	"18,19,20,21,22"	1.0	51.88	10	53.12	0	1
5	"20,21,22,23,24"	1.0	48.75	11	53.12	0	4
6	"22,23,24,25,26"	1.0	42.50	8	53.12	0	10
7	"25,26,27,28,29"	1.0	53.75	5	53.12	0	-0
8	"28,29,30,31"	1.0	51.88	9	53.12	0	1
9	"10,11,12,13,14"	1.5	54.37	5	51.25	0	-3
10	"12,13,14,15,16"	1.5	50.62	8	52.50	0	1
11	"15,16,17,18,19"	1.5	40.00	21	52.50	1	12
12	"18,19,20,21,22"	1.5	41.25	13	53.12	0	11
13	"20,21,22,23,24"	1.5	48.12	6	52.50	0	4
14	"22,23,24,25,26"	1.5	40.00	7	53.12	0	13
15	"25,26,27,28,29"	1.5	48.75	11	53.75	0	5
16	"28,29,30,31"	1.5	35.62	14	53.75	0	18
17	"10,11,12,13,14"	2.0	51.25	6	50.62	0	-0
18	"12,13,14,15,16"	2.0	51.25	11	51.88	0	0
19	"15,16,17,18,19"	2.0	47.50	10	54.37	0	6
20	"18,19,20,21,22"	2.0	52.50	8	55.00	0	2
21	"20,21,22,23,24"	2.0	41.88	12	52.50	0	10
22	"22,23,24,25,26"	2.0	45.00	11	53.12	0	8
23	"25,26,27,28,29"	2.0	47.50	12	53.75	0	6
24	"28,29,30,31"	2.0	46.88	10	53.75	0	6
25	"10,11,12,13,14"	2.5	40.00	10	49.38	0	9
26	"12,13,14,15,16"	2.5	50.00	14	49.38	0	-0

Table A.1 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
27	"15,16,17,18,19"	2.5	45.00	8	55.62	0	10
28	"18,19,20,21,22"	2.5	46.25	12	55.00	0	8
29	"20,21,22,23,24"	2.5	47.50	11	53.12	0	5
30	"22,23,24,25,26"	2.5	56.25	7	53.12	0	-3
31	"25,26,27,28,29"	2.5	58.13	11	53.75	0	-4
32	"28,29,30,31"	2.5	45.00	8	53.75	0	8
33	"10,11,12,13,14"	3.0	47.50	10	50.00	0	2
34	"12,13,14,15,16"	3.0	51.88	8	50.00	0	-1
35	"15,16,17,18,19"	3.0	48.12	7	56.25	0	8
36	"18,19,20,21,22"	3.0	53.75	11	55.00	1	1
37	"20,21,22,23,24"	3.0	49.38	6	52.50	0	3
38	"22,23,24,25,26"	3.0	49.38	12	53.12	0	3
39	"25,26,27,28,29"	3.0	43.75	11	53.75	0	10
40	"28,29,30,31"	3.0	42.50	9	53.75	0	11

A.2.2 Qwen-7B on HumanEval

Table A.2: Full Results: Qwen/Qwen2.5-Coder-7B-Instruct on HumanEval [cite: 2]

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"11,12,13,14,15"	1.0	65.00	0	68.75	0	3
2	"14,15,16,17,18"	1.0	66.25	0	68.75	0	2
3	"16,17,18,19,20"	1.0	64.38	0	68.75	0	4
4	"19,20,21,22,23"	1.0	65.62	0	68.75	0	3
5	"22,23,24,25,26"	1.0	70.62	0	68.75	0	-1
6	"23,24,25,26,27"	1.0	66.25	0	68.75	0	2
7	"8,9,10,11,12"	1.0	71.88	0	70.00	0	-1
8	"11,12,13,14,15"	1.5	64.38	0	68.75	0	4
9	"14,15,16,17,18"	1.5	63.12	0	70.00	0	6
10	"16,17,18,19,20"	1.5	65.62	0	69.38	0	3
11	"19,20,21,22,23"	1.5	66.88	0	68.75	0	1
12	"22,23,24,25,26"	1.5	61.25	0	69.38	0	8
13	"23,24,25,26,27"	1.5	62.50	0	68.75	0	6

Table A.2 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
14	"8,9,10,11,12"	1.5	68.12	0	68.75	0	0
15	"11,12,13,14,15"	2.0	59.38	0	68.12	0	8
16	"14,15,16,17,18"	2.0	64.38	0	70.00	0	5
17	"16,17,18,19,20"	2.0	63.75	0	69.38	0	5
18	"19,20,21,22,23"	2.0	58.13	0	68.75	0	10
19	"22,23,24,25,26"	2.0	62.50	0	69.38	0	6
20	"23,24,25,26,27"	2.0	61.25	0	69.38	0	8
21	"8,9,10,11,12"	2.0	70.00	0	68.12	0	-1
22	"11,12,13,14,15"	2.5	68.75	0	66.25	0	-2
23	"14,15,16,17,18"	2.5	68.75	0	70.00	0	1
24	"16,17,18,19,20"	2.5	67.50	0	69.38	0	1
25	"19,20,21,22,23"	2.5	61.88	0	69.38	0	7
26	"22,23,24,25,26"	2.5	61.88	0	69.38	0	7
27	"23,24,25,26,27"	2.5	63.75	0	69.38	0	5
28	"8,9,10,11,12"	2.5	61.25	0	68.12	0	6
29	"11,12,13,14,15"	3.0	61.25	0	65.62	0	4
30	"14,15,16,17,18"	3.0	65.62	0	68.75	0	3
31	"16,17,18,19,20"	3.0	68.12	0	68.75	0	0
32	"19,20,21,22,23"	3.0	63.75	0	69.38	0	5
33	"22,23,24,25,26"	3.0	67.50	0	69.38	0	1
34	"23,24,25,26,27"	3.0	61.88	0	69.38	0	7
35	"8,9,10,11,12"	3.0	65.00	0	68.12	0	3

A.2.3 Qwen-14B on HumanEval

Table A.3: Full Results: Qwen/Qwen2.5-Coder-14B-Instruct on HumanEval [cite: 3]

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"14,15,16,17,18"	1.0	66.25	0	66.88	0	0
2	"16,17,18,19,20"	1.0	61.88	0	66.88	0	5
3	"21,22,23,24,25"	1.0	66.88	0	66.88	0	0
4	"26,27,28,29,30"	1.0	67.50	0	68.12	0	0
5	"30,31,32,33,34"	1.0	63.75	0	68.12	0	4

Table A.3 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
6	"35,36,37,38,39"	1.0	68.12	0	68.12	0	0
7	"40,41,42,43,44"	1.0	66.25	0	68.12	0	1
8	"43,44,45,46,47"	1.0	63.12	0	68.12	0	5
9	"14,15,16,17,18"	1.5	66.88	0	66.88	0	0
10	"16,17,18,19,20"	1.5	66.25	0	67.50	0	1
11	"21,22,23,24,25"	1.5	67.50	0	66.88	0	-0
12	"26,27,28,29,30"	1.5	63.12	0	68.12	0	5
13	"30,31,32,33,34"	1.5	71.25	0	68.12	0	-3
14	"35,36,37,38,39"	1.5	65.00	0	68.12	0	3
15	"40,41,42,43,44"	1.5	65.00	0	68.12	0	3
16	"43,44,45,46,47"	1.5	60.62	0	68.12	0	7
17	"14,15,16,17,18"	2.0	65.00	0	67.50	0	2
18	"16,17,18,19,20"	2.0	61.88	0	66.88	0	5
19	"21,22,23,24,25"	2.0	60.62	0	66.88	0	6
20	"26,27,28,29,30"	2.0	65.00	0	67.50	0	2
21	"30,31,32,33,34"	2.0	64.38	0	68.75	0	4
22	"35,36,37,38,39"	2.0	64.38	0	68.12	0	3
23	"40,41,42,43,44"	2.0	66.25	0	68.12	0	1
24	"43,44,45,46,47"	2.0	65.62	0	68.12	0	2
25	"14,15,16,17,18"	2.5	64.38	0	66.88	0	2
26	"16,17,18,19,20"	2.5	62.50	0	66.88	0	4
27	"21,22,23,24,25"	2.5	62.50	0	68.12	0	5
28	"26,27,28,29,30"	2.5	66.88	0	67.50	0	0
29	"30,31,32,33,34"	2.5	63.75	0	68.75	0	5
30	"35,36,37,38,39"	2.5	70.62	0	68.12	0	-2
31	"40,41,42,43,44"	2.5	65.62	0	68.12	0	2
32	"43,44,45,46,47"	2.5	61.88	0	68.12	0	6
33	"14,15,16,17,18"	3.0	65.00	0	66.88	0	1
34	"16,17,18,19,20"	3.0	66.88	0	66.25	0	-0
35	"21,22,23,24,25"	3.0	65.00	0	68.75	0	3
36	"26,27,28,29,30"	3.0	69.38	0	66.88	0	-2
37	"30,31,32,33,34"	3.0	63.75	0	68.75	0	5
38	"35,36,37,38,39"	3.0	61.88	0	68.75	0	6
39	"40,41,42,43,44"	3.0	63.75	0	68.12	0	4

Table A.3 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
40	"43,44,45,46,47"	3.0	64.38	0	68.12	0	3

A.2.4 CodeLlama-7B on Defects4J

Table A.4: Full Results: codellama/CodeLlama-7b-Instruct-hf on Defects4J

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"10,11,12,13,14"	1.0	29.03	210	46.19	15	17
2	"13,14,15,16,17"	1.0	30.97	201	46.90	8	15
3	"16,17,18,19,20"	1.0	31.86	207	47.43	3	15
4	"19,20,21,22,23"	1.0	32.92	199	47.79	2	14
5	"22,23,24,25,26"	1.0	28.85	220	47.26	2	18
6	"25,26,27,28,29"	1.0	27.08	229	47.79	2	20
7	"27,28,29,30,31"	1.0	31.33	217	46.90	3	15
8	"9,10,11,12,13"	1.0	30.09	196	46.55	10	16
9	"10,11,12,13,14"	1.5	31.33	203	45.66	17	14
10	"13,14,15,16,17"	1.5	33.10	220	46.90	11	13
11	"16,17,18,19,20"	1.5	30.80	207	47.43	5	16
12	"19,20,21,22,23"	1.5	26.73	228	47.96	3	21
13	"22,23,24,25,26"	1.5	30.80	225	46.90	3	16
14	"25,26,27,28,29"	1.5	31.68	217	47.26	3	15
15	"27,28,29,30,31"	1.5	29.38	231	47.43	3	18
16	"9,10,11,12,13"	1.5	27.79	204	46.90	12	19
17	"10,11,12,13,14"	2.0	27.26	204	44.96	9	17
18	"13,14,15,16,17"	2.0	29.20	206	46.37	10	17
19	"16,17,18,19,20"	2.0	30.44	218	47.61	7	17
20	"19,20,21,22,23"	2.0	30.97	215	47.43	4	16
21	"22,23,24,25,26"	2.0	28.85	206	47.43	4	18
22	"25,26,27,28,29"	2.0	29.38	226	47.08	3	17
23	"27,28,29,30,31"	2.0	32.39	194	47.26	4	14
24	"9,10,11,12,13"	2.0	33.81	205	46.73	14	12
25	"10,11,12,13,14"	2.5	31.68	217	44.42	1	12
26	"13,14,15,16,17"	2.5	29.56	214	46.73	5	17

Table A.4 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
27	"16,17,18,19,20"	2.5	27.61	228	48.14	8	20
28	"19,20,21,22,23"	2.5	30.62	212	47.26	3	16
29	"22,23,24,25,26"	2.5	32.04	222	47.43	5	15
30	"25,26,27,28,29"	2.5	28.14	211	47.43	4	19
31	"27,28,29,30,31"	2.5	30.09	213	47.61	3	17
32	"9,10,11,12,13"	2.5	28.50	232	46.19	10	17
33	"10,11,12,13,14"	3.0	34.34	193	45.31	3	10
34	"13,14,15,16,17"	3.0	28.14	221	47.08	6	18
35	"16,17,18,19,20"	3.0	30.27	212	47.79	9	17
36	"19,20,21,22,23"	3.0	31.15	207	47.79	4	16
37	"22,23,24,25,26"	3.0	30.62	207	47.96	5	17
38	"25,26,27,28,29"	3.0	31.50	219	47.79	4	16
39	"27,28,29,30,31"	3.0	29.03	209	47.96	3	18
40	"9,10,11,12,13"	3.0	31.50	207	46.19	4	14

A.2.5 Qwen-7B on Defects4J

Table A.5: Full Results: Qwen/Qwen2.5-Coder-7B-Instruct on Defects4J [cite: 5]

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"11,12,13,14,15"	1.0	62.65	1	62.30	0	-0
2	"14,15,16,17,18"	1.0	59.12	0	62.30	0	3
3	"16,17,18,19,20"	1.0	59.47	0	63.36	0	3
4	"19,20,21,22,23"	1.0	62.83	0	63.19	0	0
5	"22,23,24,25,26"	1.0	61.42	0	63.01	0	1
6	"23,24,25,26,27"	1.0	61.59	0	63.54	0	1
7	"8,9,10,11,12"	1.0	62.83	0	63.19	0	0
8	"11,12,13,14,15"	1.5	61.24	0	61.95	0	0
9	"14,15,16,17,18"	1.5	62.30	0	61.42	0	-0
10	"16,17,18,19,20"	1.5	61.77	0	62.83	0	1
11	"19,20,21,22,23"	1.5	60.18	0	63.01	0	2
12	"22,23,24,25,26"	1.5	61.24	0	63.19	0	1
13	"23,24,25,26,27"	1.5	61.95	0	63.54	0	1

Table A.5 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
14	"8,9,10,11,12"	1.5	59.12	0	63.01	0	3
15	"11,12,13,14,15"	2.0	63.01	0	61.59	0	-1
16	"14,15,16,17,18"	2.0	61.24	1	61.59	0	0
17	"16,17,18,19,20"	2.0	60.71	0	62.30	0	1
18	"19,20,21,22,23"	2.0	60.88	0	63.01	0	2
19	"22,23,24,25,26"	2.0	63.36	0	63.19	0	-0
20	"23,24,25,26,27"	2.0	61.42	0	63.36	0	1
21	"8,9,10,11,12"	2.0	58.58	0	63.01	0	4
22	"11,12,13,14,15"	2.5	60.18	0	61.06	0	0
23	"14,15,16,17,18"	2.5	58.23	0	61.95	0	3
24	"16,17,18,19,20"	2.5	60.00	0	62.30	0	2
25	"19,20,21,22,23"	2.5	61.59	0	63.19	0	1
26	"22,23,24,25,26"	2.5	58.58	0	63.19	0	4
27	"23,24,25,26,27"	2.5	60.18	0	63.19	0	3
28	"8,9,10,11,12"	2.5	57.70	1	62.48	0	4
29	"11,12,13,14,15"	3.0	59.47	0	60.35	0	0
30	"14,15,16,17,18"	3.0	61.06	0	61.95	0	0
31	"16,17,18,19,20"	3.0	63.72	0	61.59	0	-2
32	"19,20,21,22,23"	3.0	60.00	0	63.19	0	3
33	"22,23,24,25,26"	3.0	59.47	0	63.36	0	3
34	"23,24,25,26,27"	3.0	62.30	0	63.54	0	1
35	"8,9,10,11,12"	3.0	60.88	0	61.06	0	0

A.2.6 Qwen-14B on Defects4J

Table A.6: Full Results: Qwen/Qwen2.5-Coder-14B-Instruct on Defects4J [cite: 6]

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
1	"14,15,16,17,18"	1.0	64.60	0	65.13	0	0
2	"16,17,18,19,20"	1.0	65.49	0	64.78	0	-0
3	"21,22,23,24,25"	1.0	65.49	1	64.96	0	-0
4	"26,27,28,29,30"	1.0	62.83	0	64.96	0	2
5	"30,31,32,33,34"	1.0	64.25	0	64.96	0	0

Table A.6 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
6	"35,36,37,38,39"	1.0	64.96	0	65.31	0	0
7	"40,41,42,43,44"	1.0	62.83	0	65.31	0	2
8	"43,44,45,46,47"	1.0	60.88	0	65.31	0	4
9	"14,15,16,17,18"	1.5	63.72	0	65.13	0	1
10	"16,17,18,19,20"	1.5	63.19	0	64.96	0	1
11	"21,22,23,24,25"	1.5	61.95	0	65.31	0	3
12	"26,27,28,29,30"	1.5	63.54	0	65.13	0	1
13	"30,31,32,33,34"	1.5	63.19	0	64.96	0	1
14	"35,36,37,38,39"	1.5	62.83	0	65.31	0	2
15	"40,41,42,43,44"	1.5	63.72	0	65.31	0	1
16	"43,44,45,46,47"	1.5	63.36	0	65.31	0	1
17	"14,15,16,17,18"	2.0	63.19	0	65.13	0	1
18	"16,17,18,19,20"	2.0	62.12	0	64.78	0	2
19	"21,22,23,24,25"	2.0	62.83	0	65.13	0	2
20	"26,27,28,29,30"	2.0	61.77	0	64.78	0	3
21	"30,31,32,33,34"	2.0	63.19	0	64.78	0	1
22	"35,36,37,38,39"	2.0	63.54	0	65.31	0	1
23	"40,41,42,43,44"	2.0	64.07	0	65.31	0	1
24	"43,44,45,46,47"	2.0	62.12	0	65.31	0	3
25	"14,15,16,17,18"	2.5	63.72	0	65.13	0	1
26	"16,17,18,19,20"	2.5	63.89	0	64.96	0	1
27	"21,22,23,24,25"	2.5	63.89	0	65.13	0	1
28	"26,27,28,29,30"	2.5	65.13	0	64.96	0	-0
29	"30,31,32,33,34"	2.5	60.00	0	64.96	0	4
30	"35,36,37,38,39"	2.5	62.65	0	65.31	0	2
31	"40,41,42,43,44"	2.5	64.78	0	65.31	0	0
32	"43,44,45,46,47"	2.5	64.78	0	65.31	0	0
33	"14,15,16,17,18"	3.0	64.96	0	65.13	0	0
34	"16,17,18,19,20"	3.0	63.36	0	64.78	0	1
35	"21,22,23,24,25"	3.0	63.54	1	65.31	0	1
36	"26,27,28,29,30"	3.0	64.96	0	64.78	0	-0
37	"30,31,32,33,34"	3.0	63.01	1	64.96	0	1
38	"35,36,37,38,39"	3.0	64.78	0	65.31	0	0
39	"40,41,42,43,44"	3.0	63.89	0	65.31	0	1

Table A.6 – *Continued*

run_id	layers	strength	base_acc	base_inv	steered_acc	steered_inv	improveme
40	"43,44,45,46,47"	3.0	64.96	0	65.13	0	0

Bibliography

[1] J. Bai and et al. Qwen-72b and qwen-14b: A series of large language models. *arXiv preprint arXiv:2309.16641*, 2023.

[2] N. Elhage and et al. Superposition, linear representations, and linear regression. *Anthropic Research*, 2022.

[3] B. Roziere and et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[4] A. Sharma and et al. Representation engineering: A top-down approach to ai control. *arXiv preprint arXiv:2310.01405*, 2023.