

Which Neurons Detect Malicious Code? A Probing Study of LLM Security Knowledge

Lam D. Dao ✉ 

RMIT University Vietnam, Hanoi, Vietnam

Vang T. Nguyen ✉ 

Hanoi University, Vietnam

Anh M. T. Bui¹ ✉ 

SOICT, Hanoi University of Science and Technology, Vietnam

Phuong T. Nguyen ✉ 

Università degli studi dell'Aquila, Italy

Abstract

Background. Large language models (LLMs) have become increasingly capable of understanding and generating source code, leading to their widespread adoption in software engineering tasks such as code completion, repair, and vulnerability detection. However, despite their strong empirical performance, the internal mechanisms through which LLMs recognize malicious or vulnerable code patterns remain poorly understood.

Aim. We investigated where the malware detection behavior is encoded inside LLMs Feed Forward Network (FFN) neurons and verified the attribution with causal interventions on the neurons identified. This aims to identify the most important neurons in detecting malicious code.

Methods. We applied mechanistic interpretability methods to locate the neurons being responsible for malware-detection behavior in three instruction-tuned LLMs: **Llama3.1-8B-Instruct**, **Mistral-v0.3-7B-Instruct**, and **Qwen2.5-7B-Instruct**. Using 1,500 malicious and 1,500 benign PyPI packages from the PyPI Malregistry, we attribute the behavior to a set of neurons.

Results. The experimental results reveal that amplifying facilitating neurons for malware detection while suppressing inhibiting ones can boost accuracy, while the reverse collapses predictions toward a single class, although the magnitude and consistency is heavily model-dependent. We demonstrated that the guardrail detection mechanism varies across models, each represents its malware detection behavior differently within its FFN layers.

Conclusions. Probing the neurons associated with security-relevant knowledge helps us gain insights into how LLMs encode malicious programming concepts, identify potentially harmful memorized behaviors, paving the way toward more reliable defense mechanisms, such as neuron-level editing, selective unlearning, and security-aware alignment for code-focused LLMs.

2012 ACM Subject Classification Security and privacy → Software security engineering

Keywords and phrases Malicious code, Unlearning methods, LLMs

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.70

Category Emerging Results, Vision & Reflection Track Paper

Related Version *Full Version:* <https://arxiv.org/pdf/2607.10221>

Funding This paper has been supported by the MOSAICO project (Management, Orchestration and Supervision of AI-agent COmmunities for reliable AI in software engineering) that has received funding from the European Union under the Horizon Research and Innovation Action (Grant Agreement No. 101189664).

* Corresponding author



© Lam D. Dao, Vang T. Nguyen, Anh M. T. Bui, and Phuong T. Nguyen;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;
Article No. 70; pp. 70:1–70:14



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Large Language Models (LLMs) have transformed software engineering (SE) by powering advanced code-generation tools such as GitHub Copilot, or DeepSeek-Coder [5, 18]. While these models significantly boost developer productivity, they also introduce serious security risks [28, 29]. Despite their strong empirical performance, the internal mechanisms through which LLMs recognize malicious or vulnerable code patterns remain poorly understood. This lack of interpretability raises important concerns regarding trustworthiness, robustness, and security, particularly in safety-critical software development scenarios where incorrect or manipulated predictions may introduce severe risks [19, 32]. Such vulnerabilities stem from the models' tendency to memorize and reproduce harmful patterns present in their pre-training or fine-tuning data, especially from public repositories and security-related datasets [2, 16]. To mitigate this, current defense strategies primarily rely on prompt engineering, safety fine-tuning, or post-hoc filtering. However, these approaches treat the model as a black box and often fail in adaptive or adversarial prompts [20, 25].

Motivation. Recent studies have shown that safety mechanisms and refusal behaviors are not uniformly distributed across the model but are encoded in specific internal representations [9]. There has been evidence showing that LLMs can memorize insecure coding patterns, generate vulnerable code, and remain susceptible to adversarial manipulation [1, 23]. Despite this insight, there is limited understanding of which layers are primarily responsible for detecting and blocking malicious code. Thus, studying which neurons and representations contribute to malicious code detection is crucial for improving transparency and enabling fine-grained control over model behavior.

Objectives. Our work aims to bridge such a gap by systematically probing the internal neurons of code LLMs to identify those that most effectively detect malicious code. Using benign and malicious PyPI packages [12], we attribute malware-detection behavior to individual feed-forward neurons and verify the attribution through causal interventions. Having pinpointed where and how LLMs recognize malicious code is a prerequisite for both future safeguards and halting the malware generation of such requests.

Methods and Findings. Experiments on `Llama3.1-8B-Instruct`, `Qwen2.5-7B-Instruct`, and `Mistralv0.3-7B-Instruct` reveal that guardrail activation is highly layer-dependent, i.e., middle layers for `Llama3.1-8B-Instruct` and `Mistralv0.3-7B-Instruct`, late layers for `Qwen2.5-7B-Instruct`. By localizing guardrail mechanisms, our work provides practical benefits: (1) enabling lightweight, early-exit safety checks that significantly reduce inference cost while maintaining high detection accuracy; (2) supporting targeted model editing and layer-specific unlearning to strengthen defenses without degrading general code-generation capability; and (3) laying the foundation for self-monitoring defenses that let a model recognize and reject malicious generation at its source.

Contributions. Through this preliminary study as a NIER paper, we advocate for fundamentally new research directions in probing and understanding LLMs for software engineering tasks. Our work represents an initial step toward systematically investigating the behaviors of LLMs in the detection of malicious code, making the following contributions.

- A practical approach to understand LLMs' behaviors in recognizing malicious code.
- An empirical study with three LLMs on real-world datasets to validate the performance.
- A replication package including code and data to foster open science [7].

Structure. Section 2 presents the background and motivate related to our work. Afterward, Section 3 describes our methodology, covering dataset construction, neuron attribution methods, and intervention conditions. Section 4 details the experimental setup. Section 5 reports and discusses the results addressing our research questions. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Motivation and Background

Recent work suggests that high-level behaviors are not evenly distributed across an LLM but rather concentrate in specific layers or sets of neurons. Di Palma et al. [9] have analyzed sentiment encoding across layers and scales using probe classifiers, identifying layers and pooling methods that best capture sentiment signals. The work showed that sentiment information is most concentrated in the mid-layers for the binary polarity task, although the analysis stops at the layer level.

Integrated Gradients (IG), proposed by Sundararajan et al. [24], provides a way to measure how much each input contributes to a prediction by accumulating gradients along a path from a baseline. Neuron Conductance (NC) [8] extends further, tracking how much of that contribution flows through a single neuron. Both apply naturally to FFN blocks, which act as key-value memories in which each neuron responds to a pattern in input and writes to the output distribution as stated by Geva et al. [11]. Dai et al. [6] used IG on those activations to trace individual facts to single neurons; Chen et al. [4] extended the idea to gated FFNs, using multiple-choice prompting as a proxy task; and Li et al. [14] applied it at the task level in NeuronLLM, a framework inspired by biological functional antagonism, identifying *good* and *bad* FFN neurons that jointly govern task performance in LLMs and confirm their roles by amplifying one group while suppressing the other, reporting the neurons it finds sit predominantly in the middle and the top layers.

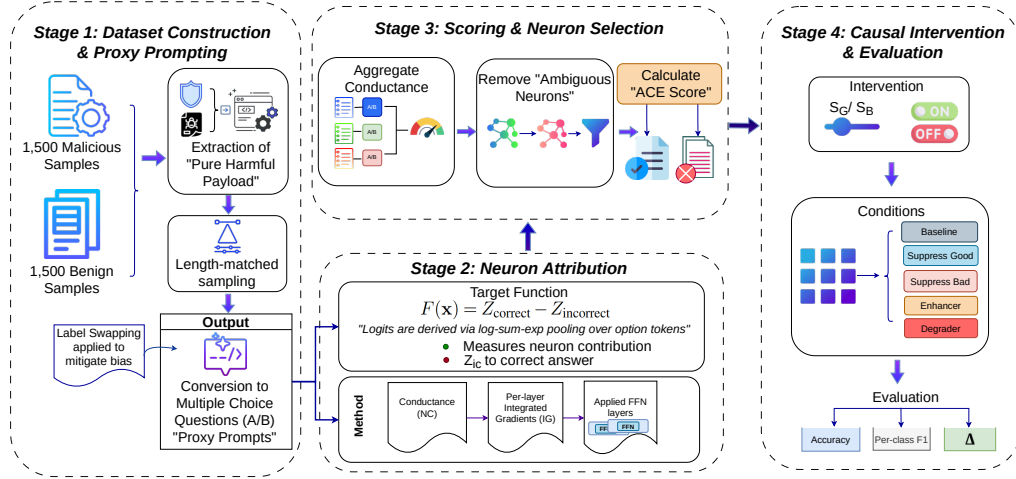
However, current interpretability-based safety research has focused predominantly on natural language jailbreaks [31, 3], often treating safety as a monolithic “refusal.” The application of these mechanistic techniques to security-relevant tasks-and to the detection of subtle, cross-file malicious logic in code-specialized LLMs in particular-remains largely unexplored [26, 13]. This leaves a critical gap in understanding whether malicious code is encoded in distinct “*toxic neurons*” or emerges from a fragile, distributed representation that current defense-in-depth layers fail to capture.

To our knowledge, this work is the first one to tackle the issue of probing LLMs for malicious code detection.

3 Methodology

We employed various mechanistic interpretability methods [8, 14] to pinpoint the neurons that support and hinder the model’s ability to detect malicious code. Both malware and benign samples were collected to perform contrastive analysis of the models’ internal representations when detecting malware versus benign code. As whown in Fig. 1, our pipeline consists of four main stages, i.e., (1) Dataset Construction & Proxy Prompting; (2) Neuron Attribution; (3) Scoring & Neuron Selection; and (4) Causal Intervention & Evaluation, explained as follows.

We use the terms “good” and “bad” neurons to refer to neurons that help or hinder the model in the malware-benign code classification task. For each example, we use attribution methods to assign a score to each neuron based on its contribution to the model’s answer. A positive score means the neuron supports the correct answer, while a negative score means



■ **Figure 1** Framework overview: Neuron-level detection of malicious code patterns.

it opposes it. After filtering and selection, the neurons that most strongly supports correct answers across both malware and benign samples form the good set, while those that most strongly work against form a bad set. These two sets of neurons are treated as task-level neurons for binary malware-benign code classification.

3.1 Dataset Construction and Proxy Prompting

Malicious PyPI packages leverage Python automation mechanisms at two key stages: install-time and import-time [12]. We developed a heuristic to collect the pure harmful payload, avoiding unwanted benign implementation in those malware packages. In particular, instead of throwing all package files into the malware samples, which may lead to false positives due to the cloning implementation, we scan for the setup file (`setup.py`), which is run during installation. For runtime attacks, we look for the initial file (often named `__init__.py`) that is suspiciously long, which may indicate it contains the malicious payload. For benign samples, to create a suitable contrastive set with the malware, we develop a heuristic to prioritize collecting package setup code files with a similar structure and length.

From the collected dataset (cf. Table 1), we ask LLMs if a code snippet is malware or benign in the form of multiple choice question A/B. To avoid the tendency of LLMs to pick the first answer or default to specific tokens [21, 30], we designed the proxy question to swap the label of A/B and change the prompt for the proxy questions.

■ **Table 1** Dataset distribution by line-count bucket.

Bucket (lines)	Malware	Benign	Total
≤ 20	98	98	196
21 – 50	1,107	1,107	2,214
51 – 100	110	110	220
101 – 200	139	139	278
> 200	46	46	92
Total	1,500	1,500	3,000

3.2 Neuron Attribution

Integrated gradients (IG) [24] is an axiomatic attribution method that assigns contributions to inputs by accumulating the gradients along an interpolation path from the baseline input x' to the actual input x as expressed in the following formula. Let $F : \mathbb{R}^n \rightarrow [0, 1]$ be the function representing a deep neural network, $\mathbf{x} \in \mathbb{R}^n$ be the input, $\mathbf{x}' \in \mathbb{R}^n$ be the baseline input. The integrated gradients calculate the attribution of each input entry by integrating the gradients along the interpolation path from x' to x .

$$\text{IntegratedGrads}_i(x) := (x_i - x'_i) \times \int_{\alpha=0}^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

The target function F is defined as: $F(x) = Z_{\text{correct}} - Z_{\text{incorrect}}$, in which Z_{correct} is the raw logit at the output layer corresponding to the correct answer's token prior to the softmax activation; and $Z_{\text{incorrect}}$ is the raw logit value corresponding to the incorrect answer. By applying the target function $F(x)$ as given, we can equally reward a neuron for amplifying the correct answer, i.e., increasing Z_{correct} and penalizing the wrong answer $Z_{\text{incorrect}}$.

Two methods [8, 14] are applied to identify neurons responsible for malware detection.

- **Per-layer IG on FFN neurons (IG).** Our second attribution method builds on the framework proposed by Li et al. [14]. Particularly, for w_i^l is the i -th neuron in a l -th layer in an FFN, $\hat{w}_i^l$ is the neuron activation value. The integrated gradient for neuron i in layer l is:

$$\text{IG}(\hat{w}_i^l) = \frac{\hat{w}_i^l}{m} \sum_{k=1}^m \frac{\partial F(\frac{k}{m} \cdot \hat{w}_i^l)}{\partial \hat{w}_i^l}$$

- **Neuron Conductance (NC).** The Neuron Conductance method [8] extends Integrated Gradients to attribute a model's output to individual hidden neurons by measuring how much signal that neuron conducts from the input to the output:

$$\text{Cond}_i^y(x) = (x_i - x'_i) \cdot \int_{\alpha=0}^1 \frac{\partial F(x' + \alpha(x - x'))}{\partial y} \cdot \frac{\partial y}{\partial x_i} d\alpha$$

3.3 Scoring and Neuron Selection

From the given methods and framework, we adopted them to our malware detection task. We calculate the Example Score by summing the conductance on all 3 questions:

$$\text{ES}_{e_j} = \sum_{t=1}^3 \text{Cond}(w, p_t^{(j)})$$

in which, the $p_t^{(j)}$ is the t -th prompt for the j -th sample. From the Example Scores values, we picked 2 sets of neurons for each sample: (i) G_j : z neurons with the highest ES; (ii) B_j : z neurons with the lowest ES. After that, we filter out the ambiguous neurons, i.e., those that are considered a Facilitator in one sample but an Inhibitor in another. Formally, a neuron (l, i) , the i -th neuron in the l -th layer, is considered as ambiguous if

$$(l, i) \in \bigcup_j G_j \text{ and } (l, i) \in \bigcup_j B_j$$

Finally, we calculate the aggregated conductance score by aggregating per-example example scores, with the ambiguous neurons masked out.

$$\text{ACE}(w) = \frac{1}{N} \sum_{j=1}^N 1[w \in G_j \cup B_j] \cdot 1[w \notin \text{Ambiguous}] \cdot ES_{e_j}(w)$$

The final set of neurons is selected: (i) G_T : top K neurons with the highest ACE score; and (ii) B_T : top K neurons with the lowest ACE score, K was set to 100.

3.4 Causal Intervention

After selecting the good and bad neurons using both methods, we intervened the neurons' values to test whether they were functional and had effects on the models' performance. We evaluated five intervention conditions on each model. Writing s_G and s_B for the multiplicative scales applied to good and bad neurons respectively, there are the following configurations: **Baseline** ($s_G=s_B=1$, no modification), **Suppress Good** ($s_G=0$, $s_B=1$), **Suppress Bad** ($s_G=1$, $s_B=0$), **Enhancer** ($s_G=2$, $s_B=0$), and **Degrader** ($s_G=0$, $s_B=2$) [14].

4 Experimental Setup

4.1 Evaluation Metrics

We evaluate model performance on a held-out balanced test set under five intervention conditions. Let TP , FP , TN , and FN denote true positives, false positives, true negatives, and false negatives, respectively, where the positive class is *malware*, then the metrics are defined as follows.

- **Per-class Precision, Recall, and F1-score.** For each class $c \in \{\text{benign}, \text{malware}\}$:

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c} \quad \text{Recall}_c = \frac{TP_c}{TP_c + FN_c} \quad F1_c = \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

- **Overall Accuracy.** Accuracy and its 95% Wilson confidence interval [17, 22, 27] are:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad \text{CI}_{95} = \frac{\hat{p} + \frac{z^2}{2n} \pm z \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where $\hat{p}$ is the observed accuracy, n is the test-set size, and $z = 1.96$.

- **McNemar's Test.** To assess whether two intervention conditions produce significantly different error patterns, we apply McNemar's test with Edwards' continuity correction [10, 15] (paired, $\alpha = 0.05$). Let b and c denote the number of samples where condition 1 is correct and condition 2 is wrong, and vice versa. The test statistic is:

$$\chi^2 = \frac{(|b - c| - 1)^2}{b + c}$$

which is asymptotically distributed as χ^2 with one degree of freedom under the null hypothesis that both conditions have equal error rates.

- **Effectiveness Gap.** To summarize the overall utility of the identified neuron sets in a single scalar, we define the effectiveness gap as: $\Delta = \text{Accuracy}_{\text{Enhancer}} - \text{Accuracy}_{\text{Degrader}}$. A large positive Δ indicates that the identified neurons are both sufficient to boost correct predictions (Enhancer) and necessary to maintain them (Degrader), providing strong causal evidence for their role in malware detection.

4.2 Model Settings

- **Model Configuration and Infrastructure.** All models are loaded in full `bfloat16` precision *without* quantization—a hard requirement, as quantization introduces numerical noise that corrupts the gradient signals needed for both IG and Conductance attribution. Models are deployed with automatic device mapping across available GPUs. `Llama3.1-8B-Instruct` consists of 8B parameters with 32 layers, while `Mistralv0.3-7B-Instruct` and `Qwen2.5-7B-Instruct` both have 7B parameters, with 32 and 28 layers, respectively.
- **Attribution Hyperparameters.** For IG, the $m = 16$ integration steps are approximated via a right Riemann sum (excluding $\alpha = 0$); since IG patches one layer at a time, the total forward-pass cost per sample is $m \times L$ (e.g., $16 \times 32 = 512$ for 32-layer models). Conductance interpolates input embeddings with the same step budget but runs all layers freely, requiring only m forward-backward passes total - substantially cheaper than per-layer IG. The attribution sample count $|\mathcal{D}_{\text{attrib}}|$ is swept over $\{10, 20, 50, 100, 200\}$ to assess sensitivity; 20 is the default used in all primary results.
- **Prompting and Truncation.** Source code samples are truncated to a maximum of 2,000 characters; samples exceeding this limit receive a `\n...[truncated]` suffix. The truncated code is embedded in a structured binary-choice prompt (Section 3). Model predictions are derived from logit argmax over the space-prefixed token sets $\{" A", " B"\}$, aggregated via log-sum-exp pooling to handle multiple tokenization variants of each option letter. This strategy is applied identically at both attribution and evaluation time, ensuring that the target function F optimized during attribution matches the criterion used to measure accuracy.
- **Data Split.** The length-matched dataset was partitioned by both class label and line-count bucket, yielding a held out test set of 2,890 samples (1,445 benign, 1,445 malware) that is used exclusively for intervention evaluation and is separated from the 110 samples used for neuron attribution and ACE scoring.

5 Results and Discussion

5.1 RQ₁: Are the probing mechanisms able to identify the most defensive neurons?

Causal interventions on the identified neurons produce measurable shifts in accuracy and per-class F1, with effect sizes differing substantially by model and method.

■ **Table 2** Intervention accuracy and Effectiveness Gap Δ . 95% Wilson CIs are within ± 0.019 of every point estimate (tightest, ± 0.009 , at the near-ceiling rows). “n.s.” = not significant at $\alpha = 0.05$; bold p -values are significant.

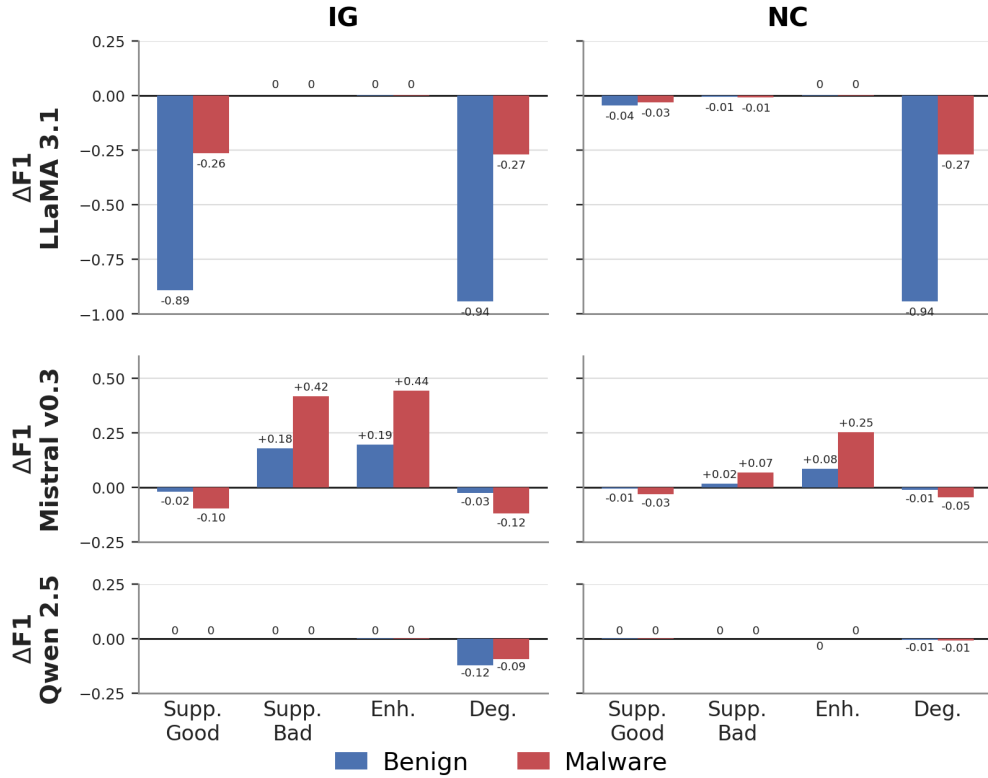
Model	Method	Base	Sup-G	Sup-B	Enh	Deg	Δ	McNemar
Llama3.1-8B-Instruct	IG	0.939	0.513	0.939	0.941	0.500	0.441	0.386 n.s.
Llama3.1-8B-Instruct	NC	0.939	0.901	0.932	0.941	0.500	0.441	0.359 n.s.
Mistralv0.3-7B-Instruct	IG	0.665	0.625	0.920	0.941	0.616	0.325	<0.001
Mistralv0.3-7B-Instruct	NC	0.665	0.652	0.696	0.800	0.645	0.154	<0.001
Qwen2.5-7B-Instruct	IG	0.939	0.939	0.940	0.941	0.832	0.109	0.181 n.s.
Qwen2.5-7B-Instruct	NC	0.939	0.941	0.939	0.939	0.932	0.007	0.617 n.s.

▷ **Intervention effects on accuracy.** Effect sizes varied across models, i.e., the high-baseline models `Llama3.1-8B-Instruct` and `Qwen2.5-7B-Instruct` (93.9%) show little response to corrective interventions (Suppress Bad, Enhancer), whereas `Mistralv0.3-7B-Instruct`

improves from 66.5% to 94.1% under the IG Enhancer. IG Suppress Bad alone raises it to 92.0%, suggesting that these inhibitory neurons drive **Mistralv0.3-7B-Instruct**’s low baseline. NC shows the same direction but weaker, with its Enhancer reaching only 80%.

Under destructive interventions (Suppress Good, Degradar), **Llama3.1-8B-Instruct** and **Qwen2.5-7B-Instruct** diverge: the former collapses to 50.0% under the Degradar (both methods) and 51.3% under IG Suppress Good, proving its identified neurons are causally necessary. The latter resists with only the IG Degradar has a mild effect (83.2%). **Mistralv0.3-7B-Instruct** degrades slightly, bounded by its already-low baseline.

IG and NC agreed on direction but not on magnitude: identical on **Llama3.1-8B-Instruct**’s Degradar ($\Delta = 0.441$ for both) but divergent sharply on Suppress Good (IG: 51.3% and NC: 90.1%). The effectiveness gap Δ_{IG}/Δ_{NC} is roughly 2 times on **Mistralv0.3-7B-Instruct** (0.325 vs 0.154) and over 10 times on **Qwen2.5-7B-Instruct** (0.109 vs 0.007). IG produces a gap of equal size or larger in every row of the table.



■ **Figure 2** Per-class $\Delta F1$ relative to baseline for each combination. Blue bars = benign class; red bars = malware class. Positive values = improvement; negative values = degradation.

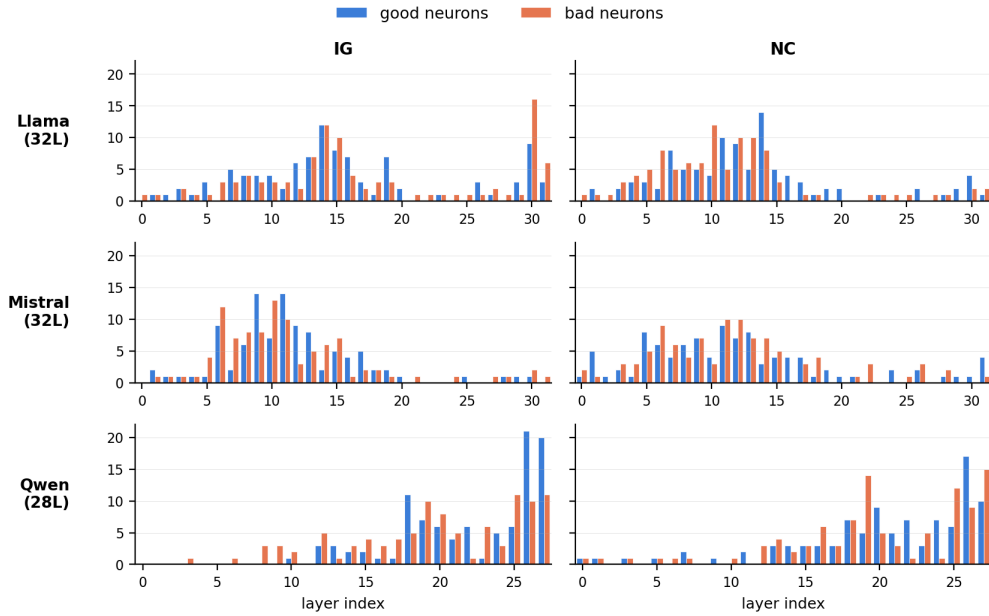
► **Intervention effects on Per-class F1.** As shown in Fig. 2, the models behave differently to the interventions. Those with strong initial baselines, such as **Llama3.1-8B-Instruct** and **Qwen2.5-7B-Instruct**, respond minimally to Suppress Bad and Enhancer, both yield $\Delta F1 \approx 0$. **Mistralv0.3-7B-Instruct**, whose baseline was asymmetric (benign 0.99, malware recall 0.33), responds more clearly: IG Suppress Bad and the Enhancer raise malware recall to 0.84–0.89 while holding benign recall at 0.99.

The destructive interventions, such as Suppress Good and Degradar, separate Llama3.1-8B-Instruct and Qwen2.5-7B-Instruct, both of which were low responders to the corrective interventions. Llama3.1-8B-Instruct collapses under both IG and NC Degradar ($\Delta F1_{\text{benign}} = -0.94$, $\Delta F1_{\text{malware}} = -0.27$) and IG Suppress Good ($\Delta F1_{\text{benign}} = -0.89$, $\Delta F1_{\text{malware}} = -0.26$), whereas Qwen2.5-7B-Instruct stays stable with only slight degradation under IG Degradar. Mistralv0.3-7B-Instruct also degrades slightly with its $\Delta F1_{\text{benign}} = -0.03$, $\Delta F1_{\text{malware}} = -0.12$ under the IG Degradar. NC showed less visible effects than IG overall: dramatic only on Llama3.1-8B-Instruct’s Degradar, and weaker across other models and intervention types.

Answer to RQ₁. Both IG and NC identify neurons that causally encode malware-detection behavior, though effect sizes are strongly model-dependent. Mistralv0.3-7B-Instruct suppresses inhibiting neurons alone increases the accuracy. For Llama3.1-8B-Instruct and Qwen2.5-7B-Instruct, the Enhancer yields negligible gains, yet the Degradar collapses Llama3.1-8B-Instruct to 50.0%. Meanwhile, Qwen2.5-7B-Instruct is relatively robust and stable to these intervention effects. Overall, IG achieves larger or equal effectiveness gaps compared to NC.

5.2 RQ₂: How are the defensive neurons distributed across the layers and how stable is their selection across runs?

We investigate where in the network these neurons reside and whether attribution consistently points to the same regions across different data subsets.



■ **Figure 3** Per-layer distribution of top-100 good (blue) and bad (orange) neurons.

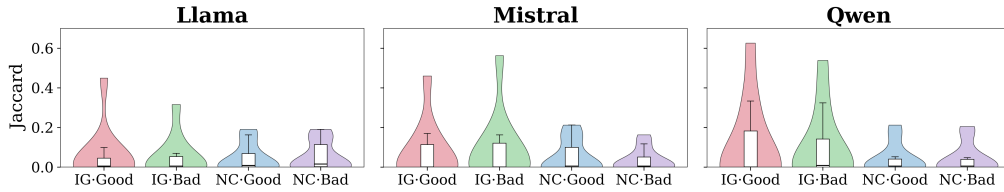
▷ **Distribution.** The results are shown in Fig. 3. Llama3.1-8B-Instruct (32 layers): Both IG and NC place the primary concentration in the middle layers. Under IG, good neurons peak in Layer 13, and bad neurons peak in Layers 14-15, with an additional spike in Layer 30. NC shows a similar mid-layer peak at Layer 14 with good and bad neurons co-located throughout Layers 6-18, plus a modest late spike at Layer 30.

Mistralv0.3-7B-Instruct (32 layers): The concentration is the tightest among all three models. Both IG and Conductance place good and bad neurons almost exclusively in layers 5-14, with IG peaking at layer 10 and Conductance still within the same band. After layer 15, both methods select almost no neurons, and the IG and Conductance distributions are most similar across all three models.

Qwen2.5-7B-Instruct (28 layers): The dominant signal is in the late layers. Using IG, the distribution shows a secondary cluster around layer 12, followed by a sharp spike at layers 25-26, peaking at layer 26, the highest single-layer count across all models. Under NC, the signal is more spread across layers 10-27, with the same peak at layer 26 but less abrupt than IG. The first ten layers carry almost no signal under either method.

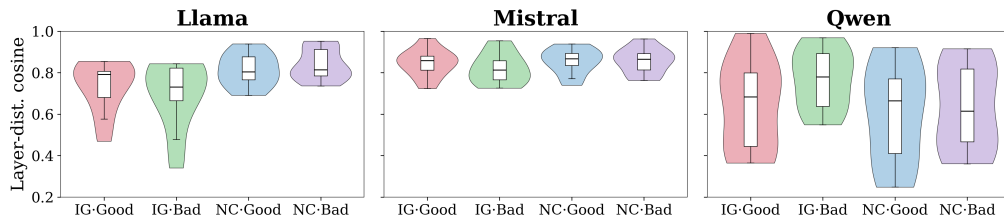
Each model exhibits a clear preferred layer region: middle layers for **Llama3.1-8B-Instruct** and **Mistralv0.3-7B-Instruct**, late layers for **Qwen2.5-7B-Instruct**. A consistent pattern across all three models is that good and bad neurons co-localize in the same layer bands rather than being separated into different regions of the network.

▷ **Stability.** We test whether the neuron selection is consistent by repeating attribution five times on different random subsets of the data ($K = 100$ neurons per run, seeds 0-4). We compute two complementary similarities over the 10 pairs of runs: Jaccard similarity of the selected neuron sets, capturing whether the same individual neurons are picked; and cosine similarity of the per-layer neuron-count histograms.



■ **Figure 4** Pairwise Jaccard similarity of the top-100 neuron sets across 5 stability runs.

As shown in Fig. 4, Jaccard values are low across all models and methods (all medians below 0.20), showing that the specific neurons identified do not reliably repeat across attribution subsets. **Llama3.1-8B-Instruct** IG is the least consistent (medians ≈ 0.03 - 0.04 for both classes, upper tails to ≈ 0.45), with NC slightly more stable (≈ 0.10). **Mistral-v0.3-7B-Instruct** IG has higher medians (≈ 0.10 good, ≈ 0.12 bad) and a wide upper tail for bad neurons (≈ 0.57). **Qwen2.5-7B-Instruct** IG-Good has the highest median of all conditions (≈ 0.17 , tail to 0.62), reflecting the sharp single-layer concentration in Fig. 3, whereas **Qwen2.5-7B-Instruct** NC collapses to near-zero (≈ 0.03).



■ **Figure 5** Pairwise cosine similarity of the per-layer neuron-count histograms across the same 5 runs. Medians span between 0.6 and 0.9, an order of magnitude above the corresponding Jaccards.

As shown in Fig. 5, despite low Jaccard values, the layer-level shape of the selection is substantially more stable: cosine values across all conditions range from 0.61 to 0.86, an order of magnitude above the corresponding Jaccard. **Mistralv0.3-7B-Instruct** is the most stable at the layer level. All four conditions (IG/NC $\times$ Good/Bad) have medians of 0.82-0.86 with narrow IQRs (≈ 0.78 -0.90), so the early-to-middle band (layers 5-14) is consistently identified regardless of which neurons within it are selected. **Llama3.1-8B-Instruct** is intermediate, with NC more stable than IG: NC-Good (0.813 ± 0.076) and NC-Bad (0.840 ± 0.073) are tighter than IG-Good (0.732 ± 0.120) and IG-Bad (0.694 ± 0.159). IG-Bad shows the widest spread (outliers to ≈ 0.45) because the late layer-30 spike is not always selected. **Qwen2.5-7B-Instruct** is the least stable, having the widest IQRs of all three models (especially NC-Good, 0.606 ± 0.231 , run pairs ranging cosine ≈ 0.4 -0.8): IG concentrates sharply at layers 25-26 while NC spreads across the final third.

Answer to RQ₂. Each model concentrates its malware-detection signal in a unique layer band: middle layers for **Llama3.1-8B-Instruct** and **Mistralv0.3-7B-Instruct**, late layers for **Qwen2.5-7B-Instruct**. The selected neurons are not stable across runs. However, stability resides at the layer level. **Llama3.1-8B-Instruct** and **Mistralv0.3-7B-Instruct**, which exhibit this layer stability behavior, also respond most clearly to causal intervention effects in RQ1, whereas **Qwen2.5-7B-Instruct**, whose layer distribution is the most dispersed, responds the least, a co-occurrence suggesting that stable layer-level localization may be a prerequisite for effective causal interventions.

5.3 Discussion

- **Implications.** By probing the neurons associated with security-relevant knowledge, we can gain deeper insights into how LLMs internally encode malicious programming concepts and insecure coding patterns. Such understanding enables the identification of potentially harmful memorized behaviors and paves the way toward more reliable defense mechanisms, including neuron-level editing, selective machine unlearning, and security-aware alignment for code-focused LLMs. Moreover, this line of research opens opportunities for fine-grained security interventions at the neuron and layer levels, such as targeted suppression of harmful behaviors, activation steering, and layer-specific hardening strategies. For instance, understanding which layers are more strongly associated with malicious code recognition may help explain why certain adversarial attacks succeed, e.g., by bypassing safety-relevant intermediate representations, and may inspire the integration of dedicated safety adapters.
- **Limitations.** Due to limited computational resources, we were able to conduct experiments with relatively small LLMs, i.e., less than 8B parameters. Being a NIER contribution, *the primary objective of our work is to introduce and demonstrate the feasibility of a novel probing-based approach for understanding malicious code detection in LLMs, rather than to provide a large-scale or exhaustive evaluation*. Essentially, since the proposed probing methodology does not require expensive retraining or architecture modifications, it can be scaled to larger and more capable code-generation models with a lot more layers in a relatively lightweight and model-agnostic manner. We believe that the encouraging preliminary results motivate further investigation and broader empirical validation in future work.

5.4 Threats to Validity

- **Internal Validity.** Attribution identifies a layer-level signal that is stable across seeds, but the specific neurons selected are not: pairwise Jaccard similarity remains below 0.20 (Fig. 4). Our conclusions, therefore, hold at the granularity of layer bands rather than at the level of individual neurons. Attribution hyperparameters ($m=16$, $K=100$, $|\mathcal{D}_{\text{attrib}}|=20$) are also fixed. Hence, a sensitivity analysis is left to future work.
- **External Validity.** Our findings are derived exclusively from three 7-8B LLMs and Python/PyPI packages, focusing on install-time and import-time payloads. They may not transfer to larger or differently aligned models, to other ecosystems (e.g., npm, Maven) or attack vectors (e.g., typosquatting), or to malware more recent than the PyPI Malregistry [12].

6 Conclusion and Future Work

This paper proposes a practical approach to locate the FFN neurons responsible for distinguishing malicious from benign code inside three instruction-tuned LLMs, i.e., **Llama3.1-8B-Instruct**, **Mistralv0.3-7B-Instruct**, **Qwen2.5-7B-Instruct**. Using two attribution methods, we attributed the detection behavior to a specific set of neurons and verified the attribution using causal interventions. Furthermore, we found that each model concentrates its signal in a distinct layer band, and that neuron localization is stable at the layer level rather than at the individual neuron level. This work provides a foundation for LLM safety behavior in recognizing malicious code, offers open avenues for early-exit safety checks, and enables targeted model editing. For future work, we plan to bridge code recognition and generative refusal, disentangle surface-level cues from semantic intent, and extend the analysis to larger models and other software ecosystems.

Data Availability Statement

The replication package of our work is available in a public repository [7]. It includes the source code, data, and instructions in the form of a detailed README.MD file for reproducing the results obtained in this paper.

References

- 1 Ahmet Emir Bosnak, Sahand Moslemi, Mayasah Lami, and Anil Koyuncu. Explicit vulnerability generation with llms: An investigation beyond adversarial attacks. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2025, Auckland, New Zealand, September 7-12, 2025*, pages 821–826. IEEE, 2025. doi:10.1109/ICSME64153.2025.00086.
- 2 Nicholas Carlini, Florian Tramèr, et al. Extracting training data from large language models. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2633–2650. USENIX Association, 2021. URL: <https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting>.
- 3 Jianhui Chen, Xiaozhi Wang, Zijun Yao, Yushi Bai, Lei Hou, and Juanzi Li. Finding safety neurons in large language models. *arXiv e-prints*, pages arXiv-2406, 2024.
- 4 Lihu Chen, Adam Dejl, and Francesca Toni. Identifying query-relevant neurons in large language models for long-form texts. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI)*, pages 23595–23604, 2025. doi:10.1609/AAAI.V39I22.34529.
- 5 Mark Chen, Jerry Tworek, et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. arXiv:2107.03374.

- 6 Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.acl-long.581.
- 7 Lam D. Dao, Vang T. Nguyen, Anh M. T. Bui, and Phuong T. Nguyen. Replication Package for “Which Neurons Detect Malicious Code? A Probing Study of LLM Security Knowledge”. <https://github.com/dd10620/esem-neuron-package/>, 2026. Accessed: 2026-05-27.
- 8 Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. How important is a neuron? *CoRR*, abs/1805.12233, 2018. arXiv:1805.12233.
- 9 Dario Di Palma, Alessandro De Bellis, Giovanni Servedio, Vito Walter Anelli, Fedelucio Narducci, and Tommaso Di Noia. LLaMAs have feelings too: Unveiling sentiment and emotion representations in LLaMA models through probing. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6124–6142, Vienna, Austria, July 2025. Association for Computational Linguistics. doi:10.18653/v1/2025.acl-long.306.
- 10 Allen L. Edwards. Note on the “correction for continuity” in testing the significance of the difference between correlated proportions. *Psychometrika*, 13(3):185–187, 1948. doi:10.1007/BF02289261.
- 11 Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.emnlp-main.446.
- 12 Wenbo Guo, Zhengzi Xu, Chengwei Liu, Cheng Huang, Yong Fang, and Yang Liu. An empirical study of malicious code in pypi ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 166–177. IEEE, 2023. doi:10.1109/ASE56229.2023.00135.
- 13 Aftab Hussain, Md Rafiqul Islam Rabin, and Mohammad Amin Alipour. On trojan signatures in large language models of code. *arXiv preprint*, 2024. arXiv:2402.16896, doi:10.48550/arXiv.2402.16896.
- 14 Wenjie Li, Guansong Pang, Hezhe Qiao, Debin Gao, and David Lo. Identifying good and bad neurons for task-level controllable llms, 2026. doi:10.48550/arXiv.2601.04548.
- 15 Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi:10.1007/BF02295996.
- 16 Milad Nasr, Nicholas Carlini, et al. Scalable extraction of training data from (production) language models. *CoRR*, abs/2311.17035, 2023. doi:10.48550/arXiv.2311.17035.
- 17 Robert G Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in medicine*, 17(8):857–872, 1998.
- 18 Ipek Ozkaya. Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Softw.*, 40(3):4–8, May 2023. doi:10.1109/MS.2023.3248401.
- 19 Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot’s code contributions. *Commun. ACM*, 68(2):96–105, January 2025. doi:10.1145/3610721.
- 20 Ethan Perez, Saffron Huang, H. Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3419–3448. Association for Computational Linguistics, 2022. doi:10.18653/v1/2022.EMNLP-MAIN.225.

- 21 Pouya Pezeshkpour and Estevam Hruschka. Large language models sensitivity to the order of options in multiple-choice questions. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2006–2017, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.findings-naacl.130.
- 22 Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009. doi:10.1016/J.IPM.2009.03.002.
- 23 Angelica Spina, Marco Russodivito, Simone Scalabrino, and Rocco Oliveto. Peeking inside the black box: Training data exposure in code language models. *Journal of Systems and Software*, 234:112729, 2026. doi:10.1016/j.jss.2025.112729.
- 24 Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17, pages 3319–3328. JMLR.org, 2017. URL: <http://proceedings.mlr.press/v70/sundararajan17a.html>.
- 25 Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing NLP. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2153–2162. Association for Computational Linguistics, 2019. doi:10.18653/V1/D19-1221.
- 26 Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in neural information processing systems*, 36:80079–80110, 2023.
- 27 Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.
- 28 Zhou Yang, Jieke Shi, Junda He, and David Lo. Natural attack for pre-trained models of code. In *Proceedings of the 44th International Conference on Software Engineering, ICSE ’22*, pages 1482–1493, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3510003.3510146.
- 29 Zhou Yang, Bowen Xu, Jie M. Zhang, Hong Jin Kang, Jieke Shi, Junda He, and David Lo. Stealthy backdoor attack for code models. *IEEE Transactions on Software Engineering*, 50(4):721–741, 2024. doi:10.1109/TSE.2024.3361661.
- 30 Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *The Twelfth International Conference on Learning Representations*, 2024. URL: <https://openreview.net/forum?id=shr9PXz7T0>.
- 31 Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint*, 2023. arXiv:2310.01405.
- 32 Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *CoRR*, abs/2307.15043, 2023. doi:10.48550/arXiv.2307.15043.