# Exploring *Coding Spot*: Understanding Parametric Contributions to LLM Coding Performance

**Anonymous ACL submission**

## Abstract

Large Language Models (LLMs) demonstrate strong code generation and comprehension abilities, yet the extent to which different programming languages are processed independently or within a shared parametric space remains unclear. Inspired by cognitive neuroscience, we introduce Coding Spot, a specialized parametric region that facilitates coding capacity in LLMs. Our findings show that targeted modifications to this subset significantly affect coding performance while largely preserving non-coding functionalities, suggesting that LLMs exhibit parametric specialization similar to function-specific brain regions. This indicates that coding knowledge may not be uniformly distributed across the model but instead concentrated in distinct regions that play a crucial role in task-specific performance. Enhancing our understanding of how LLMs internalize coding knowledge offers new directions for optimizing model architectures and improving code-related applications.

## 1 Introduction

Large Language Models (LLMs) have initiated a significant transformation in computational code processing, showcasing advanced capabilities in tasks such as code generation and comprehension across a wide range of programming languages (Chen et al., 2021; Austin et al., 2021; Li et al., 2022). Models such as Llama 3 (Dubey et al., 2024), GPT-4o (Achiam et al., 2023), and Claude 3.5 Sonnet (Anthropic, 2024) have achieved considerable success, establishing themselves as essential tools for automating programming tasks and enhancing developer productivity.

Despite these successes, a fundamental question remains: how do these models internally represent and organize the coding knowledge necessary for such tasks? More specifically, it is unclear whether the knowledge required for programming tasks is uniformly distributed across the model's parameters or if certain parameter subsets exhibit specialization for coding-related functionalities. This question is reminiscent of findings in cognitive science, where specific brain regions, such as Broca's and Wernicke's areas, are specialized for language processing (Broca et al., 1861; Wernicke, 1874). Inspired by this analogy, we hypothesize that LLMs may similarly exhibit task-specific parametric regions, particularly those dedicated to coding tasks.

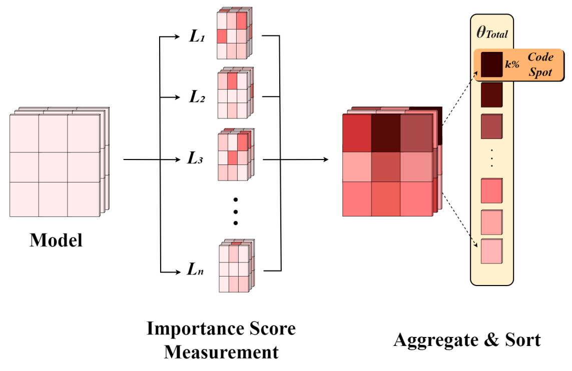

Figure 1: Overview of the framework for extracting and analyzing Coding Spot within LLMs. The process begins with the model undergoing importance scoring independently for n programming languages. The Importance Scores extracted from each language are aggregated for each parameter and then sorted in descending order. The parameters within the top k% of Importance Scores are defined as the Coding Spot

In this study, we introduce the concept of the *Coding Spot*, a theoretical construct representing a subset of parameters within LLMs that are particularly critical for code-related capabilities. Analogous to domain-specific regions in the brain, the *Coding Spot* embodies a specialized parametric region that is crucial for the model's proficiency in coding tasks. By identifying and analyzing these critical regions, we aim to provide new insights into the internal parametric architecture of LLMs and their ability to manage coding and general tasks.

Our primary objective is to conduct a rigorous examination of the parametric structure of LLMs, with a focus on uncovering the role of the *Coding Spot*. We evaluate the consequences of modifying this subset of parameters on both coding and non-coding tasks, shedding light on the compartmentalized nature of the LLMs' internal architecture. This investigation offers a deeper understanding of how LLMs handle domain-specific knowledge and draws compelling parallels to the cognitive specialization observed in the human brain.

## 2 Coding Spot

Our methodology aims to identify and analyze the *Coding Spot*, a specialized subset of parameters crucial for coding proficiency in LLMs.

**Methodological Framework** The core of our methodology is a systematic algorithm designed to identify critical subsets of parameters—termed the *Coding Spot*—from a large pool of LLM parameters. By employing parameter importance scoring, our framework efficiently isolates the parameters most relevant to coding tasks, drawing analogies to the specialization observed in the human brain, where distinct regions are responsible for different cognitive functions.

**Parameter Importance Scoring** To isolate the *Coding Spot*, we begin by fine-tuning LLMs on datasets containing code from individual programming languages. The purpose of this fine-tuning is not to build a new task-specific model but to extract accurate parameter gradients via backpropagation. These gradients allow us to construct language-specific parameter subsets that are vital for coding tasks.

Given a dataset $D_l$ corresponding to a programming language $l$ and a set of parameters $\theta = [\theta_1, \theta_2, \ldots, \theta_d]$, our goal is to estimate how changes in the loss function $L(D_l, \theta)$ relate to each parameter. Using the first-order Taylor expansion, the model's loss in response to a specific parameter $\theta_j$ can be approximated as:

$$L(D_l, \theta) \approx L(D_l, \theta|_{\theta_j=0}) + \frac{\partial L(D_l, \theta)}{\partial \theta_j} \cdot \theta_j \quad (1)$$

This equation highlights how the loss is affected by parameter $\theta_j$, informed by its gradient during fine-tuning. The parameter importance score $I_j^l(\theta)$, which quantifies each parameter's contribution to coding tasks, is computed as:

$$I_j^l(\theta) \approx \left| \frac{\partial L(D_l, \theta)}{\partial \theta_j} \right| \cdot |\theta_j| \quad (2)$$

This score provides a direct measure of each parameter's relevance in the context of a specific programming language $l$, revealing the most critical parameters for coding tasks. The role of the fine-tuned model here is solely to facilitate precise gradient extraction, not to be used in subsequent analyses.

**Aggregating Parametric Importance Across Diverse Languages** Once we calculate the importance scores $I_j^l(\theta)$ for each parameter within individual languages, the next step is to aggregate these scores across multiple languages. For each parameter $\theta_j$, we compute a total importance score $I_j^{\text{total}}(\theta)$ by summing the importance scores across all languages in the set $L$:

$$I_j^{\text{total}}(\theta) = \sum_{l \in L} I_j^l(\theta) \quad (3)$$

This aggregation captures the global importance of each parameter, allowing us to identify parameters that consistently influence coding tasks across diverse languages. By sorting the parameters in descending order based on their total importance scores, we isolate a subset of parameters—referred to as the *Coding Spot*—that are crucial for coding proficiency.

**Defining the *Coding Spot*** The *Coding Spot* is identified by selecting the top $k\%$ parameters from the sorted list. These parameters, which consistently demonstrate high importance across multiple languages, form a concentrated subset responsible for coding tasks. The value of $k$ is empirically determined to ensure that we capture the most critical parameters while avoiding redundancy.

## 3 Experiments

The objective of our experimental evaluation is to quantify the role and impact of the *Coding Spot* in LLMs, particularly its influence on both task-specific (e.g., coding) and general tasks (e.g., mathematical or commonsense reasoning). We systematically deactivate the identified *Coding Spot* parameters to examine their specialization and robustness, thus providing insights into the broader implications of parameter specialization in LLMs.

|  | General |  |  |  |  |  | Code |
|---|---|---|---|---|---|---|---|
|  | **GSM8K** | **HellaSwag** | **MMLU** | **TruthfulQA** | **WinoGrande** | **Avg. GTC (%)** | **HumanEval** |
| ***CodeLlama 7B Instruct*** |  |  |  |  |  |  |  |
| **Original** | 18.12 | 48.32 | 39.54 | 39.21 | 64.56 | 41.95 | 87.2 |
| ✹ **0.0025%** | 1.90 | 36.01 | 24.63 | 39.32 | 53.51 | 31.07 (-25.93%) | 22.56 (-74.13%) |
| ✹ **0.01%** | 2.27 | 35.08 | 23.99 | 36.91 | 52.41 | 30.13 (-28.17%) | 16.46 (-81.12%) |
| ✹ **0.09%** | 0.23 | 27.23 | 22.94 | 49.72 | 51.38 | 30.30 (-27.77%) | 1.83 (-97.90%) |
| ✹ **0.25%** | 0.00 | 25.85 | 22.93 | 51.52 | 50.51 | 30.16 (-28.10%) | 0.00 (-100.00%) |
| ***Llama 3.1 8B Instruct*** |  |  |  |  |  |  |  |
| **Original** | 76.72 | 59.10 | 67.96 | 54.08 | 73.64 | 66.30 | 97.56 |
| ✹ **0.0025%** | 2.35 | 44.47 | 42.32 | 43.61 | 60.14 | 38.58 (-41.81%) | 20.12 (-79.38%) |
| ✹ **0.01%** | 2.65 | 38.83 | 29.47 | 41.55 | 58.41 | 34.18 (-48.44%) | 9.76 (-90.00%) |
| ✹ **0.09%** | 0.61 | 26.82 | 23.57 | 49.58 | 49.17 | 29.95 (-54.83%) | 0.00 (-100.00%) |
| ✹ **0.25%** | 0.15 | 26.36 | 22.95 | 51.03 | 48.46 | 29.79 (-55.07%) | 0.00 (-100.00%) |
| ***Llama 3.2 3B Instruct*** |  |  |  |  |  |  |  |
| **Original** | 64.67 | 52.22 | 60.42 | 49.77 | 67.56 | 58.93 | 94.51 |
| ✹ **0.0025%** | 2.35 | 41.39 | 38.71 | 45.07 | 55.96 | 36.70 (-37.73%) | 15.24 (-83.87%) |
| ✹ **0.01%** | 2.20 | 36.31 | 30.30 | 45.66 | 53.75 | 33.64 (-42.91%) | 6.71 (-92.90%) |
| ✹ **0.09%** | 1.29 | 26.66 | 23.36 | 50.53 | 49.49 | 30.27 (-48.64%) | 0.00 (-100.00%) |
| ✹ **0.25%** | 1.21 | 26.21 | 22.94 | 49.54 | 49.96 | 29.97 (-49.14%) | 0.00 (-100.00%) |

Table 1: Performance comparison of three LLMs on general and coding tasks after deactivating a percentage of the model identified as *Coding Spot*. GTC represents General Task Change, and ✹ marks models with deactivated parameters.

## 3.1 Experimental Setup

The experimental setup was carefully designed to ensure comprehensive evaluation of both specialized and general model performance. Detailed information about the datasets, models, and evaluation benchmarks can be found in Appendix B.

**Datasets** Our study employed carefully curated training datasets and evaluation benchmarks to rigorously assess both specialized and general performance aspects of the models. For the fine-tuning phase, models were trained on the `nampdn-ai/tiny-codes` (Nam Pham, 2023) dataset, meticulously filtered to include only pure code, thus excluding any content related to instruction following or English language capabilities. This focus was crucial to exclude instruction-following and English abilities, ensuring that models could extract and focus only on coding skills.

For code-related task evaluation, we utilized the HumanEval benchmark (Chen et al., 2021), which assesses the models' capabilities in generating correct code solutions across a range of programming challenges. To evaluate general task proficiency, we employed a diverse set of benchmarks including GSM8K (Cobbe et al., 2021) (assessing mathematical reasoning), HellaSwag (Zellers et al., 2019) (for commonsense reasoning), and MMLU (Hendrycks et al., 2020) (for multi-task evaluation), TruthfulQA (Lin et al., 2021) (to assess truthfulness of responses), and WinoGrande (Sakaguchi et al., 2021) (for coreference reasoning).

**Models** We evaluated three state-of-the-art LLMs of varying sizes: CodeLlama 7B Instruct (Roziere et al., 2023), Llama 3.1 8B Instruct, and Llama 3.2 3B Instruct (Dubey et al., 2024). These models were chosen to assess how different architectures and scales influence the identification and impact of the *Coding Spot*. During fine-tuning, Python was excluded to test whether the *Coding Spot* extends its functionality beyond language-specific constraints, thereby assessing its generalization across different coding environments.

**Evaluation Metrics** The primary metric for code-related tasks was the HumanEval score, while general tasks were evaluated using accuracy metrics on GSM8K, HellaSwag, and MMLU. The key experimental procedure involved systematically nullifying varying percentages of the *Coding Spot* parameters and evaluating their impact on both types of tasks. By excluding Python during fine-tuning, we tested the hypothesis that the *Coding Spot* reflects a broader coding proficiency, generalizable across different programming languages.

## 4 Results

### 4.1 Main Results

The results clearly demonstrate the critical role of the *Coding Spot* in both task-specific and general task performance. As shown in Table 1, deactivating even a small percentage of the most crucial

| | Avg. GTC (%) | Code Change (%) | $M_s$ |
|---|---|---|---|
| | ***CodeLlama 7B Instruct*** | | |
| **Original** | 41.95 | 87.20 | - |
| ✸ **0.0025%** | -25.93% | -74.13% | 5.44 |
| ✸ **0.01%** | -28.17% | -81.12% | 5.52 |
| ✸ **0.09%** | -27.77% | -97.90% | 6.75 |
| ✸ **0.25%** | -28.10% | -100.00% | 6.82 |
| | ***Llama 3.1 8B Instruct*** | | |
| **Original** | 66.30 | 97.56 | - |
| ✸ **0.0025%** | -41.81% | -79.38% | 2.70 |
| ✸ **0.01%** | -48.44% | -90.00% | 2.65 |
| ✸ **0.09%** | -54.83% | -100.00% | 2.61 |
| ✸ **0.25%** | -55.07% | -100.00% | 2.60 |
| | ***Llama 3.2 3B Instruct*** | | |
| **Original** | 58.93 | 94.51 | - |
| ✸ **0.0025%** | -37.73% | -83.87% | 3.41 |
| ✸ **0.01%** | -42.91% | -92.90% | 3.34 |
| ✸ **0.09%** | -48.64% | -100.00% | 3.19 |
| ✸ **0.25%** | -49.14% | -100.00% | 3.15 |

Table 2: Comparison of LLM performance after *Coding Spot* parameter deactivation. GTC (%) and Code Change (%) show changes in accuracy for general and coding tasks. $M_s$ measures task-specific impact. ✸ marks models with deactivated parameters.

parameters caused a significant decline in performance across the benchmarks.

For code-related tasks, such as HumanEval, Llama 3.1 8B Instruct, which initially achieved a score of 97.56, saw its performance plummet to zero when 0.09% or 0.25% of the *Coding Spot* parameters were deactivated. This sharp decline highlights the essential role of these parameters in maintaining coding proficiency and suggests that the *Coding Spot* encapsulates critical knowledge for code generation, generalizable across different programming languages.

For general tasks, such as GSM8K (mathematical reasoning), performance similarly exhibited notable declines when *Coding Spot* parameters were deactivated. This indicates that the parameters critical for coding tasks also contribute to broader cognitive functions, underscoring the polysemantic nature of the *Coding Spot*. However, tasks such as HellaSwag (commonsense reasoning) were less affected, indicating that distinct neural components may govern different general tasks, reflecting a modular structure within LLMs.

### 4.2 Impact of *Coding Spot* on Performance

Our further exploration provides insights into the intricate dynamics between task-specific and general performance upon *Coding Spot* parameter deactivation. The monosemanticity score $M_s$, crafted to evaluate the changes in output across specialized and general tasks, is sensitive to the extent of parameter removal:

$$M_s = \frac{\Delta\text{Coding Task Performance}}{1 + \Delta\text{General Task Performance}} \qquad (4)$$

Table 2 illustrates that Llama 3.1 and 3.2 models attained their highest monosemanticity scores with a minimal deactivation (0.0025%), signifying that even a small fraction of parameter alteration can disproportionately influence coding tasks while leaving general capabilities relatively unscathed. This suggests a high density of critically functional parameters in these models, reflecting precise and efficient organization of the *Coding Spot*. Conversely, for CodeLlama, more extensive deactivation (0.25%) achieved peak scores, which may be attributed to its robust coding specialization derived from extensive training on diverse code repositories. This hints at a broader parametric allocation for coding tasks, confirming the presence of a more extensive *Coding Spot* compared to instruction-tuned models like Llama 3.1 8B.

Our findings reveal that *Coding Spot* parameters play crucial roles in tasks requiring logical and numerical reasoning. This is evident from the performance drop in the GSM8K benchmark, where mathematical reasoning—a domain overlapping with coding capabilities—is assessed. While commonsense reasoning tasks like HellaSwag remained stable, the decline in complex, math-related tasks highlights the essential nature of *Coding Spot* parameters. These parameters not only enhance coding abilities but also support high-level problem-solving tasks, emphasizing their versatile roles within LLMs across broader cognitive functions.

## 5 Conclusion

We introduced the *Coding Spot*, a specialized parameter subset in LLMs essential for both code generation and general tasks. Our experiments demonstrate that deactivating even a small percentage of these parameters leads to significant performance declines, confirming their critical role. These findings suggest that the *Coding Spot* supports multiple domains, offering valuable insights for future work on optimizing LLM architectures to enhance both task-specific and general capabilities.

## Limitations

While our study provides significant insights into the structure and function of Large Language Models (LLMs), certain limitations must be acknowledged transparently. One of the primary limitations is the empirical selection of the threshold percentage $k\%$ for identifying the *Coding Spot*. Although empirical processes are commonly utilized to approximate optimal configurations when theoretical guidance is lacking, we recognize this approach may not guarantee absolute optimality across all LLM architectures. Future research could benefit from developing more robust, mathematically grounded methods for threshold determination.

Additionally, our methodology involves nullifying the *Coding Spot* by setting the parameters to zero. Although this strategy effectively isolates the impact of these parameters, it might raise questions about interpretability, given the inherently positive and negative distribution of parameter weights. Setting them to zero provides a clear baseline for assessing their absence. However, we acknowledge this raises potential questions about non-zero alternatives, which might lead to different and perhaps unexpected model dynamics. While this alternative strategy has dividends in preserving network activity, introducing non-zero values could result in uncontrolled variance and divergent behaviors, rendering the results less interpretable. Thus, while this is an interesting avenue for future exploration, particularly for understanding robustness and sensitivity, the decision for zeroing parameters remains justified given current interpretability and controllability needs.

Our study was conducted exclusively using Llama models. This decision was intentional, designed to facilitate direct intra-architecture comparisons. A consistent model framework minimizes extraneous variability, allowing for a more focused analysis of parameter significance across different conditions. While this approach inherently limits our findings' generalizability to non-Llama architectures, it ensures robust internal comparison and serves as a strong foundation for future studies that can expand to more diverse LLM families.

While these constraints may be seen as potential shortcomings, we view them as informed choices given the current study's scope and analytical goals. They provide a baseline upon which future refinements and broader model inclusivity can be built.

## Ethics Statement

This research adheres to ethical guidelines in both the design and execution of experiments. The LLMs evaluated in this study were trained using publicly available data, and no private or sensitive information was involved. However, we acknowledge that LLMs, including those optimized for code generation, can raise concerns regarding fairness, bias, and security. It is important that future applications of this research take into account potential risks related to the misuse of automated code generation tools, especially in safety-critical contexts. We encourage further research on addressing these ethical concerns and ensuring the responsible deployment of LLM technologies.

## References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

Anthropic. 2024. Introducing claude 3.5 sonnet.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Paul Broca et al. 1861. Remarks on the seat of the faculty of articulated language, following an observation of aphemia (loss of speech). *Bulletin de la Société Anatomique*, 6(330-357):27.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. 2022. Toy models of superposition. *arXiv preprint arXiv:2209.10652*.

Jerry A. Fodor. 1983. *The Modularity of Mind*. The MIT Press, Cambridge, MA.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

Nam Pham. 2023. tiny-codes (revision c13428e).

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Carl Wernicke. 1874. *Der aphasische Symptomencomplex: eine psychologische Studie auf anatomischer Basis*. Cohn & Weigert.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

## A Related Work

Research on the coding capabilities of LLMs has gained significant attention, with models such as Llama 3 (Dubey et al., 2024), GPT-4o (Achiam et al., 2023), and Claude 3.5 Sonnet (Anthropic, 2024) demonstrating strong performance in generating code across various programming languages. These models have successfully automated programming tasks by producing syntactically correct and logically coherent code. However, the underlying mechanisms that enable such capabilities remain unclear, particularly regarding how coding knowledge is distributed and organized within the model's parameters. Understanding this parametric specialization is key to improving LLMs' ability to handle tasks such as code generation and comprehension.

LLM interpretability is a critical area of research aimed at uncovering how models process and store knowledge. Traditional techniques, such as saliency maps and gradient-based approaches (Simonyan and Zisserman, 2014; Sundararajan et al., 2017), have provided insights into neural network behavior by identifying feature importance. However, the scale and complexity of LLMs pose unique challenges in pinpointing the specific parameter subsets responsible for coding proficiency. This limitation has motivated the development of targeted methods for understanding the internal mechanisms of LLMs, especially in specialized tasks like code generation.

Our work draws inspiration from theories of modularity in both biological systems and artificial models. In cognitive science, the concept of modularity suggests that certain brain regions specialize in distinct functions, such as Broca's and Wernicke's areas for language processing (Fodor, 1983). Similarly, modularity has been proposed in neural networks, where specific neurons or regions are believed to specialize in particular tasks (Andreas et al., 2016). Building on this analogy, we introduce the *Coding Spot*, a specialized parametric region within LLMs dedicated to handling programming tasks. By identifying such parametric specialization, we aim to provide deeper insights into the internal architecture of LLMs and their proficiency in code-related tasks.

Recent advancements in parameter efficiency techniques, such as adapters (Houlsby et al., 2019) and parameter-efficient fine-tuning methods (Zaken et al., 2021), highlight the growing interest in improving model adaptability without retraining the entire network. These methods emphasize the importance of selectively fine-tuning critical parameter subsets for task-specific performance. Our work contributes to this field by offering a structured approach to identifying and optimizing task-critical parameters, specifically for coding tasks within LLMs.

Additionally, the concepts of monosemantic and polysemantic neurons (Elhage et al., 2022) have shaped our understanding of parameter specialization. Monosemantic neurons, which respond to specific stimuli, provide a framework for investigating parameter contributions within LLMs. In the context of code generation, identifying these specialized neurons or parameters is crucial for understanding task-specific organization. Our proposed *Coding Spot* builds on this concept, positing that certain parameter subsets are responsible for coding tasks while preserving general non-coding functionalities.

In summary, our research builds on existing work in LLM interpretability, modularity, and parameter efficiency. By introducing the *Coding Spot*, we bridge theoretical insights from cognitive science with practical advances in artificial model specialization, offering a novel framework for understanding and optimizing the coding capabilities of LLMs.

## B Experimental Setup

### Datasets

To evaluate the performance of our methodology, we use the `nampdn-ai/tiny-codes` (Nam Pham, 2023) dataset, which includes Bash, C#, C++, Go, Java, JavaScript, Julia, Ruby, Rust, and TypeScript. This dataset offers a diverse environment for fine-tuning LLMs on language-specific benchmarks, allowing a comprehensive evaluation of coding capabilities. The dataset includes real-world code samples for assessing both code generation and comprehension tasks within each language.

During preprocessing, non-code content was filtered out, and language-specific tokenization was applied to retain the syntactic and semantic integrity of each programming language, ensuring an accurate evaluation of the models.

It is critical to remove non-coding content, as our focus is on discerning the specific parameters that contribute to coding prowess. Inclusion of text related to instruction-following abilities or general reasoning could inadvertently skew the importance scores, misleadingly suggesting the relevance of parameters that are not genuinely integral to coding functions. This rigorous filtering guarantees that our analysis truly isolates the *Coding Spot* responsible for coding, avoiding confounding factors associated with unrelated linguistic tasks.

### Languages and Benchmarks

In our study, we selected a diverse set of programming languages—Bash, C#, C++, Go, Java, JavaScript, Julia, Ruby, Rust, and TypeScript. This selection covers a broad spectrum of programming paradigms, providing a comprehensive evaluation of the models' abilities across various coding styles and syntax.

We employed the HumanEval (Chen et al., 2021) benchmark, which is widely recognized for testing the Large Language Models' capability to generate code that is both syntactically correct and functionally coherent. This benchmark offers a structured framework for evaluating the proficiency of models in managing diverse coding tasks.

In addition to the coding benchmarks, our study included several other datasets aimed at assessing models on a variety of reasoning and task-specific abilities. The GSM8K (Cobbe et al., 2021) benchmark was used to evaluate mathematical reasoning capabilities, challenging models to solve arithmetic and logic problems effectively. HellaSwag (Zellers et al., 2019), a benchmark designed for commonsense reasoning, offered insights into the LLMs' ability to navigate narrative completion tasks with contextual understanding.

For multi-task evaluation, we integrated MMLU (Hendrycks et al., 2020), which tests models on a wide range of academic and professional subjects, offering a holistic view of their reasoning and knowledge synthesis across disciplines. TruthfulQA (Lin et al., 2021) was employed to assess the truthfulness of model responses, pushing LLMs to adhere closely to factuality and truth in their outputs. Lastly, WinoGrande (Sakaguchi et al., 2021) tested coreference reasoning, emphasizing the ability to resolve pronouns based on contextual cues, thereby evaluating linguistic and contextual coherence.

These non-coding benchmarks collectively provide a rigorous and diverse assessment framework, ensuring a comprehensive evaluation of the LLMs beyond code generation, highlighting their versatility and adaptability across different reasoning scenarios and tasks.

## Models

Our experiments are conducted using three state-of-the-art models: CodeLlama 7B Instruct (Roziere et al., 2023), Llama 3.1 8B Instruct (Dubey et al., 2024), and Llama 3.2 3B Instruct (Dubey et al., 2024). These models were selected for their distinct capabilities and architectural strengths, providing a diverse set of test subjects for our study on parametric specialization.

The CodeLlama 7B Instruct model is particularly noteworthy for its expertise in code-related tasks. It is pre-trained on a vast corpus of coding-specific data, enabling it to navigate complex coding challenges with precision and efficiency. Its architecture is fine-tuned for tasks that require generating syntactically and semantically accurate code, making it an ideal candidate for testing the *Coding Spot* hypothesis in environments demanding high-level coding proficiency.

On the other hand, Llama 3.1 8B Instruct and Llama 3.2 3B Instruct offer robust performance across a broader range of instructional content beyond pure coding, including natural language understanding and generalized reasoning. These models provide insight into how the *Coding Spot* can extend its specialized responsibilities across varied cognitive demands. Their sizeable parameter sets and nuanced training regimes allow them to adapt well to fine-tuning on language-specific datasets, providing a comprehensive view of parameter specialization and monosemantic behavior across different scales and instruction types.