How Programming Concepts and Neurons Are Shared in Code Language Models

Amir Hossein Kargaran[1] **Yihong Liu**[1] **François Yvon**[2] **Hinrich Schütze**[1]

[1]LMU Munich & Munich Center for Machine Learning
[2]Sorbonne Université & CNRS, ISIR
{amir, yihong}@cis.lmu.de

Abstract

Several studies have explored the mechanisms of large language models (LLMs) in coding tasks, but most have focused on programming languages (PLs) in a monolingual setting. In this paper, we investigate the relationship between multiple PLs and English in the concept space of LLMs. We perform a few-shot translation task on 21 PL pairs using two Llama-based models. By decoding the embeddings of intermediate layers during this task, we observe that the concept space is closer to English (including PL keywords) and assigns high probabilities to English tokens in the second half of the intermediate layers. We analyze neuron activations for 11 PLs and English, finding that while language-specific neurons are primarily concentrated in the bottom layers, those exclusive to each PL tend to appear in the top layers. For PLs that are highly aligned with multiple other PLs, identifying language-specific neurons is not feasible. These PLs also tend to have a larger keyword set than other PLs and are closer to the model's concept space regardless of the input/output PL in the translation task. Our findings provide insights into how LLMs internally represent PLs, revealing structural patterns in the model's concept space. Code is available at https://github.com/cisnlp/code-specific-neurons.

1 Introduction

Most state-of-the-art autoregressive large language models (LLMs) perform well on coding tasks (Chen et al., 2021; Hou et al., 2024; Lyu et al., 2024; DeepSeek-AI et al., 2024; Jiang et al., 2024). Including code in the pre-training data has become a common practice in LLM pre-training, even for models not specifically designed for coding (Aryabumi et al., 2024). Most of these LLMs involved in coding tasks are pre-trained on multiple programming languages (PLs) (Li et al., 2023; DeepSeek-AI et al., 2024; Jiang et al., 2024; Guo et al., 2024). This raises an intriguing question: How does pre-training on multiple PLs and English affect the behavior of the models' "concept space" in coding tasks? More specifically: **RQ1.** Does the model use English or a PL as a kind of "pivot" language? **RQ2.** Can we identify language-specific neurons for PLs and English? Do PLs and English influence one another and neurons are shared across PLs and English?

Observation	NLs	PLs
1) English detour	✓	✓(shared with PLs)
2) High alignment	✓(English)	✓(other PLs, e.g., C#)
3) English neuron ID	✗	✓
4) Non-English/PL neuron ID	✓	? (inconsistent)

Table 1: Differences between natural languages (NLs) and programming languages (PLs) in English-centric LLMs. 1) LLMs' layers reach non-English tokens through a detour via English (Wendler et al., 2024). The same occurs when outputting PLs, though English is shared with PL tokens (§3.1). 2) Non-English languages show high cross-lingual alignment with English in LLMs' intermediate layers (Kargaran et al., 2024), while PLs, including C#, exhibit high alignment with each other (§3.2). 3), 4) It is challenging to identify English-specific neurons whose ablation does not affect non-English. It is easy to find neurons specific for Non-English (e.g., French) (Tang et al., 2024). For PLs, there are English ablatable neurons with minimal performance degradation over PLs, but for some PLs (e.g., C#, see §3.3) finding ablatable neurons without affecting other PLs is hard.

As summarized in Table 1, we observe both similarities and differences in how LLMs represent natural languages versus PLs.

Contributions. To investigate the relationship between English and multiple PLs in the LLM's concept space, we apply methods from the field of interpretability. Specifically, we focus on two models from the Llama family: CodeLlama 7B (Rozière et al., 2023) and Llama 3.1 8B (Dubey et al., 2024). We use the logit lens technique (Nostal-

Findings of the Association for Computational Linguistics: ACL 2025, pages 26905–26917

gebraist, 2020), which involves applying the "unembedding" operation prematurely at intermediate, non-final layers. This approach provides insight into the model's internal numerical representations, which are otherwise difficult to interpret. We create a super-parallel dataset of 42 translation directions (21 pairs) across seven PLs using the dump of GeeksforGeeks (GeeksforGeeks, 2008) prepared by Zhu et al. (2022b). Our results show that the selected Llama models assign high probabilities and top ranks to expected tokens during translation in the last layer, meaning they completely understand the translation task. Tracking token probabilities across layers for different PLs and English using logit lens (see Figure 1), we observed: (1) Most tokens in the first half of the layers have low probabilities, near zero, across PLs and English. (2) Tokens from English and various PLs appear in the intermediate layers, mostly in the second half of the layers. (3) Most tokens belong to English, followed by all PLs. Among individual PLs, the output PL comes next, followed by PLs such as C++ and C#, which have some of the largest keyword sets. We use our super-parallel data and measure the cross-lingual alignment for these PLs and find that C# is more aligned with most languages but not in all cases. For example, the best-aligned PL for PHP and Python is JavaScript.

We also explore how neuron activations are shared across 11 PLs and English. We use language activation probability entropy (Tang et al., 2024) to identify language-specific neurons. Our analysis reveals the following insights: (1) Language-specific neurons are more concentrated in the bottom layers, followed by another smaller peak observed around the three quarter point of the layers. (2) Among language-specific neurons, those exclusive to a single PL tend to appear in the top layers. (3) For PLs such as C# and Java, which closely align with multiple other PLs, identifying language-specific neurons is challenging.

2 Materials and methods

We use three established methods to uncover the concept space of LLMs, using datasets from PLs at parallel, keyword, and raw levels.

2.1 Datasets

Super-parallel PL. Most of the parallel datasets available in code community research (Zhu et al. (2022a,b); Lachaux et al. (2020), *inter alia*) come

Figure 1: Illustration of logit lens (Nostalgebraist, 2020) applied to CodeLlama 7B for the task of translating a `for` loop from Java to Rust (showing only Rust loop here). The y-axis shows layers, the x-axis input tokens, and color next-token probabilities (red: low, blue: high). Terms decoded in intermediate layers, such as `interval`, `range`, `until`, and `ten`, are not keywords in Java or Rust but belong to other PLs (Python, Go, Ruby) and English. In Rust, the `..` syntax defines a range. The tokens `until` and `through`, which decode for the same position but with lower probabilities or in earlier layers, share similar semantics with this syntax.

from GeeksForGeeks (GeeksforGeeks, 2008), a website that hosts data structure and algorithm problems along with solutions in up to seven different PLs: C++, Java, Python, C#, JavaScript, PHP, and C. According to GeeksForGeeks, the solution programs for the same problem follow the same structure, including consistent variable names, resulting in semantic consistency across the different languages. In most cases, the programs for the same problem share the same set of comments in the same order, indicating that they are parallel at the snippet level. We use the (Zhu et al., 2022b) dump of GeeksForGeeks and create a super-parallel dataset for all seven PLs, containing 581 parallel code snippets, each available for all seven PLs. We retain only programs that are available in all PLs and that have the same number of snippets to ensure alignment across all seven PLs.

English and PL keywords. We gather, for 22 PLs, programming-specific keywords, as well as the names of other built-ins starting from (Meyer and McCulloch, 2022). For brevity, we refer to these as PL keywords. We also extract English keywords from PanLex (Kamholz et al., 2014), which contains words from several thousand languages, including English. We only keep keywords that the model's tokenizer represents as a single token and remove numbers from this list (if represented numerically). Note that PLs have a limited vocabulary consisting primarily of keywords whereas

natural languages have an extensive and continuously evolving lexicon. Additionally, many PLs are influenced by older PLs (Sebesta, 2016), leading to shared structures and common keywords like `if`, `for`, `while`, and `return`.

Raw PL and English. We take raw code of eleven popular PLs from the GitHub Code dataset (CodeParrot, 2022). It consists of 115 million code files from GitHub in 32 PLs. We select the following eleven popular PLs (GitHut, 2024): C, C++, C#, Go, HTML, Java, JavaScript, PHP, Python, Ruby, and Rust. We also use the English Wikipedia as the source for English texts. We limit each language to 50,000 code files/articles.

2.2 Models

We focus on models from the Llama family, which are autoregressive and decoder-only transformers (Vaswani et al., 2017). We focus on pre-trained models rather than fine-tuned ones with instruction tuning or RLHF to minimize confounding factors. We select two models: CodeLlama 7B (Rozière et al., 2023) and Llama 3.1 8B (Dubey et al., 2024). We choose models around 7B parameters, which are considered a base size in the LLM community. CodeLlama 7B is pre-trained on 500B tokens from a code-heavy dataset. It is first initialized with Llama 2 (Touvron et al., 2023b) model weights, which are pre-trained on general-purpose text and code. CodeLlama is the only family of Llama models introduced primarily for coding tasks. The latest models in the Llama family are general foundation models. Llama 3.1 8B is the latest model in the family with a base size around 7B parameters and is pre-trained on 15 trillion tokens of general-purpose text and code.

2.3 Method 1: Interpreting latent embeddings

Following Wendler et al. (2024), we use logit lens (Nostalgebraist, 2020) instead of tuned lens (Belrose et al., 2023) to decode intermediate embeddings, as tuned lens is trained to map internal states to the final next-token prediction, which may lose the signal of interest. We use logit lens to find which of the PLs or English is closer to the abstract concept space of the selected models.

Logit lens. A transformer model at layer ℓ can be viewed in two parts: (i) a lower part, which includes layers up to and including layer ℓ, that maps input tokens to hidden states, and (ii) an upper part, which includes layers after ℓ that convert hidden states into logits. The core idea of logit lens is to see the lower part as a complete transformer and apply $\mathbf{W_U}$, the "unembedding" matrix, to project the hidden state at layer ℓ, $\mathbf{h}^{(\ell)}$, into logit scores. These logit scores are then transformed into token probabilities via the softmax operation. The logit lens operation can be defined as:

$$\text{LogitLens}(\mathbf{h}^{(\ell)}) = \text{LayerNorm}[\mathbf{h}^{(\ell)}]\mathbf{W_U}.$$

Few-shot translation. The task is to translate the preceding PL (e.g., Java) code snippet into another PL (e.g., Python). We show the model four code snippets with their correct translations, followed by a fifth code snippet without its translation, and ask the model to predict the next tokens. With such a prompt, the model can infer that it should translate the fifth code snippet. Since the fifth predicted code snippet could diverge at some point and affect all the subsequent tokens, we predict the tokens one by one and replace the previous tokens with the expected ones. We use our super-parallel PL dataset (§2.1) for the fifth code snippet (both input and output PLs). For every input token, at each layer, we compute the probabilities of the top $\alpha = 10$ decoded tokens using logit lens and classify them as belonging to English or one or more PLs using the keywords dataset (§2.1).

As for the four-shot code snippets, we always use parallel data for basic structures, as shown in the example below (Input PL: Java, Output PL: Python).

```
Java: String message = ""; - Python: message = ""
Java: public class MyClass {} - Python: class MyClass:
Java: public int value = 5; - Python: value = 5
Java: public void doSomething() {} - Python: def do_something():
Java: for (int i = 0; i < 10; i++) - Python:
```

2.4 Method 2: Cross-lingual alignment

We employ MEXA (Kargaran et al., 2024), a measure of cross-lingual alignment, to determine which PL aligns most closely with the majority of the selected PLs in the model's intermediate layers. To compute MEXA, we generate code snippet embeddings using position-weighted averaging (Muennighoff, 2022) and assess alignment based on cosine similarity comparisons. The higher the score, the greater the alignment, with values ranging between 0 and 1.

MEXA. Given a decoder-only transformer model m, MEXA computes the cross-lingual alignment score for language L_1 relative to a pivot language L_2. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n parallel sentences (i.e., code snippets) in L_1 and

L_2. We use our super-parallel dataset (§2.1) for each pair. First, we compute sentence embeddings using model m at layer ℓ with position-weighted averaging. Given a sentence s, its corresponding embedding is denoted as $\mathbf{e}^{(\ell)}(s)$. We construct a similarity matrix $\mathbf{C}(L_1, L_2, m, \ell) \in \mathbb{R}^{n \times n}$, where each element $c_{ij}(\ell)$ represents the cosine similarity between the embeddings of sentence s_i in L_1 and sentence s_j in L_2. The diagonal elements $c_{ii}(\ell)$ correspond to the similarity between parallel sentence pairs. The MEXA alignment score for matrix $\mathbf{C}(L_1, L_2, m, \ell)$ is defined as:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\left(c_{ii}(\ell) > \max_{j \neq i}\{c_{ij}(\ell), c_{ji}(\ell)\}\right),$$

where $\mathbb{I}$ is the indicator function, which returns 1 if the condition holds and 0 otherwise. This measures how often a parallel sentence pair has the highest similarity compared to any non-parallel pairs.

2.5 Method 3: Language-specific neurons

We use language activation probability entropy (LAPE) (Tang et al., 2024), which outperforms similar methods in identifying language-specific regions across natural languages. We use LAPE to identify language-specific neurons in each model and analyze their impact on other languages.

Neurons in FFN. Llama-based models (Touvron et al., 2023a) use a transformer architecture with a GLU variant (Shazeer, 2020). Like other transformer architectures, their core building blocks include multi-head self-attention (MHA) and feed-forward networks (FFNs). Let $\tilde{\mathbf{h}}^{(\ell)}$ denote the output of the MHA module in the ℓ-th layer, computed using the previous layer's hidden states and trainable parameters. The FFN module, which outputs the hidden state $\mathbf{h}^{(\ell)} \in \mathbb{R}^{d_1}$, in a GLU variant transformer is given by:

$$\mathbf{h}^{(\ell)} = \left(\phi(\tilde{\mathbf{h}}^{(\ell)}\mathbf{W}_1^{(\ell)}) \otimes \tilde{\mathbf{h}}^{(\ell)}\mathbf{W}_3^{(\ell)}\right) \cdot \mathbf{W}_2^{(\ell)},$$

where $\mathbf{W}_1^{(\ell)}, \mathbf{W}_3^{(\ell)} \in \mathbb{R}^{d_1 \times d_2}$ and $\mathbf{W}_2^{(\ell)} \in \mathbb{R}^{d_2 \times d_1}$ are learnable parameters, and $\phi(\cdot)$ denotes the activation function. In LAPE, a *neuron* is defined as the linear transformation of a single column in $\mathbf{W}_1^{(\ell)}$ followed by the application of the non-linear activation function. Thus, each FFN module contains d_2 neurons. A neuron indexed by r in the ℓ-th FFN layer is considered "active" if its activation value $\phi(\tilde{\mathbf{h}}^{(\ell)}\mathbf{W}_1^{(\ell)})_r$ exceeds zero.

LAPE. To compute LAPE, we feed LLMs different texts, each written in a single language from raw PL and English texts (§2.1). For the r-th neuron in the ℓ-th layer, we calculate the activation probability when processing texts in language z:

$$p_{\ell,r}^{z} = \mathbb{E}\left(\mathbb{I}\big(\phi(\tilde{\mathbf{h}}^{(\ell)}\mathbf{W}_1^{(\ell)})_r > 0\big) \mid \text{language } z\right),$$

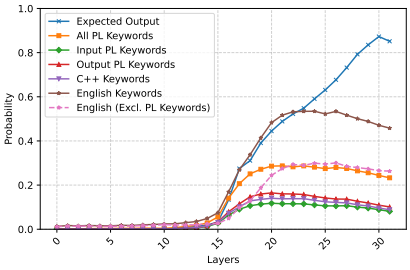
where $\mathbb{I}$ is the indicator function. This probability is empirically estimated as the likelihood that the neuron's activation value exceeds zero. We obtain the probability distribution across languages and normalize it via sum normalization to compute the normalized probability $p'^{z}_{\ell,r}$ for each language z. The entropy of this distribution is:

$$\text{LAPE}_{\ell,r} = -\sum_{z \in \mathcal{L}} p'^{z}_{\ell,r} \log(p'^{z}_{\ell,r}).$$

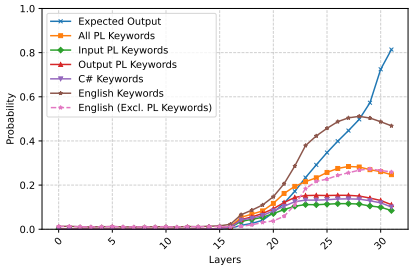
where $\mathcal{L}$ is the set of languages. We designate neurons with low LAPE scores as "language-specific neurons," as they show a predilection for activation in response to one or two languages, while showing reduced activation probabilities for others. A neuron is deemed specific to language z if its corresponding activation probability $p_{\ell,r}^{z}$ surpasses a predefined threshold.

LAPE is highly dependent on hyperparameter thresholds. The first hyperparameter is the activation threshold, set at the activation probability corresponding to the τ quantile. The default for LAPE is $\tau = 0.95$. For CodeLlama 7B/Llama 3.1 8B, this corresponds to activation probability thresholds of 0.531 and 0.554, respectively, meaning selected neurons must exhibit activation probabilities exceeding these values for at least one language. The second threshold, the filter threshold γ, retains only a small fraction of neurons as language-specific by selecting those in the bottom γ of LAPE scores. The default setting is $\gamma = 0.01$. However, since this results in varying numbers of selected neurons across languages and makes the comparison between different languages harder to interpret, we instead compute the average number of selected neurons and select the same number, ν, for each language. For the default settings of both selected models, ν is around 400.

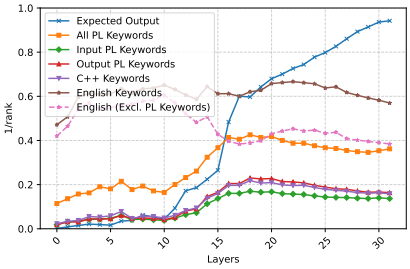
Controlled generation. To assess the impact of the selected neurons, we set their activation values to zero or zero out the corresponding parameters and then measure changes in model performance. Specifically, we compute language model perplexities (PPLs) to examine how much removing these neurons affects various languages.



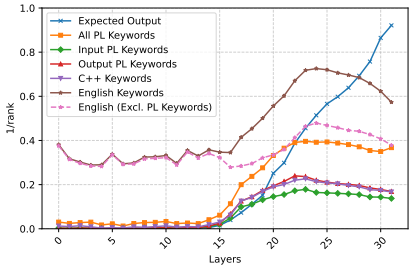
(a) Code Llama 7B; probability value.



(b) Llama 3.1 8B; probability value.



(c) Code Llama 7B; $\frac{1}{rank}$ value.



(d) Llama 3.1 8B; $\frac{1}{rank}$ value.

Figure 2: Language keyword probability or $\frac{1}{rank}$ value (best keyword rank) during translation task. The PLs contributing the most to each score, selected from the 22 PL keywords, are C++ and C#.

3 Results

3.1 Method 1: Interpreting latent embeddings

We present the results of interpreting latent embeddings for the translation task in Figure 2. Neither English nor PL keywords exhibit noticeable probability during the first half of the layers (Figures 2a, 2b). Although these probabilities remain negligible, English keywords still appear among the top rank decoded tokens in the first half of the layers (Figures 2c, 2d); this occurs much less frequently for PL keywords.

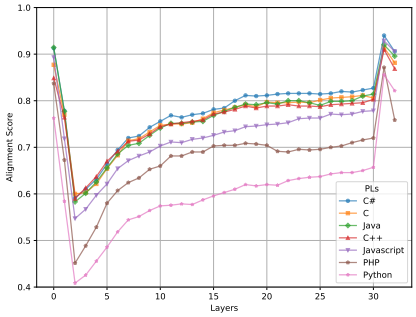
Around the half point (rougly, layer 15), the probabilities of English and PL keywords, as well as expected tokens, begin to rise sharply (Figures 2a, 2b). English and PL keywords overtake the expected tokens at first. While expected token probability continues increasing until the final layers, English and PL keyword probabilities decline, particularly when English token probability crosses over the expected token probability.

In the final layer, while the expected token holds rank = 1, English keywords (excluding PL keywords) and PL keywords maintain a high and similar $\frac{1}{rank}$ value of 0.4 each (Figures 2c, 2d), indicating their presence among the top decoded tokens. Among individual PL keywords, the output PL dominates in both rank and probability measures, followed by popular PLs like C++ and C# (which have some of the largest keyword sets), while the input PL has less influence. This distribution holds across different PL keywords: rising in the second half of the layers, peaking, and then decreasing in the final layers. Notably, many expected tokens are variable names, symbols, numbers, or punctuation, which typically fall outside the different PL keyword sets.

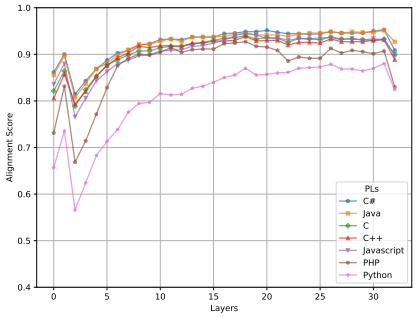
Regarding the comparison between CodeLlama 7B and Llama 3.1 8B: In terms of token probability, Llama 3.1 8B exhibits a slower initial rise in expected token probability, followed by a sharp increase in the top three layers. In contrast, CodeLlama 7B demonstrates a more gradual increase throughout. In terms of rank, CodeLlama 7B consistently shows a mainstream presence of English keywords. However, their distribution shifts: in the first half of the layers, they primarily consist of English keywords excluding PL keywords, while in the second half, they increasingly include English keywords that overlap with PL keywords. For Llama 3.1 8B, the presence of PL keywords also increases in the second half of the layers, reaching a $\frac{1}{rank}$ value of 0.7 at layer index 23 for English keywords shared with PL keywords.

3.2 Method 2: Cross-lingual alignment

We present the results of cross-lingual alignment in Figure 3. We compute alignment scores for all pairs of PLs and determine which PL aligns better with others. C# achieves the best alignment

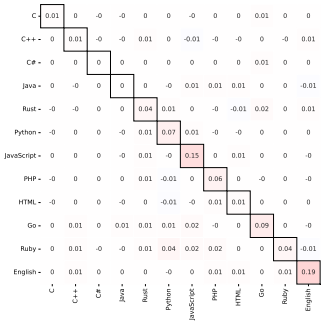


(a) Code Llama 7B

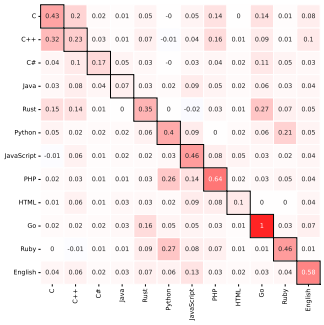


(b) Llama 3.1 8B

Figure 3: MEXA alignment score. The minimum value of the MEXA alignment score is 0. The figures are limited to scores above 0.4 for better visualization.



(a) Code Llama 7B



(b) Llama 3.1 8B

Figure 4: Impact of LAPE identification ($\nu = 400, \tau = 0.95$) on PPL increase. The element at row i, column j represents the PPL change for language j due to perturbations in the language i region.

overall across all layers in both models, though the difference between C-family PLs and Java is minimal. Both models show fewer alignments for Python. JavaScript is the best-aligned PL for both PHP and Python. The high alignment of C# and C++ further supports the influence of popular PLs, as discussed in Section 3.1. This finding is also aligned with Quan et al. (2025), who find that although Python is the most familiar language for existing LLMs and competition-level code benchmarks, model performance improves over Python when responding in C++ for most of the models, including for the instruction-tuned version of the Llama 3.1 8B model.

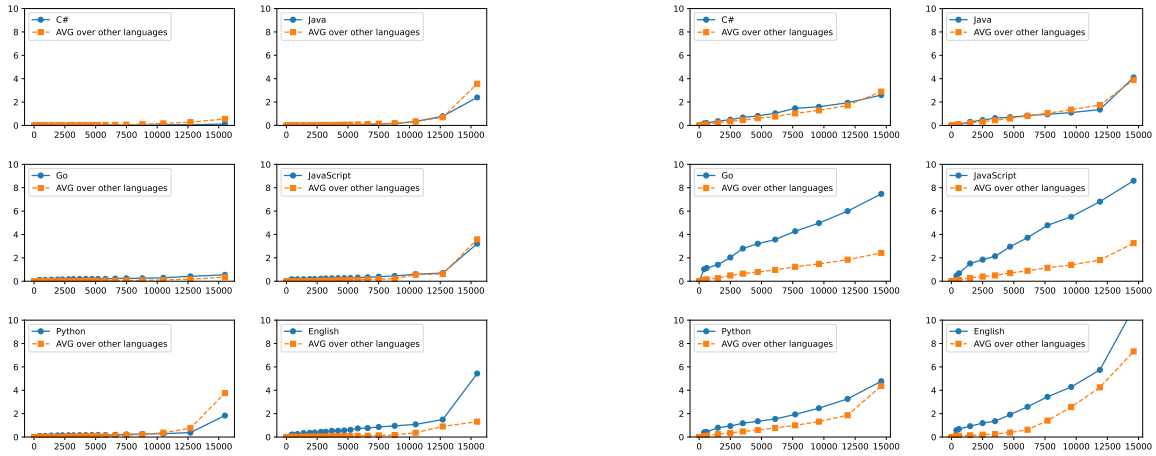
The alignment scores consistently increase except for two instances: first, in the bottom layers (layer index 2 in Figures 3a, 3b), where representations diverge from the "input space"; and second, immediately before the final layer (layer index 31 in Figures 3a, 3b), where they transition into the final "output space". The alignment of different PLs, especially in the layer preceding the final layer, indicates the high quality of the parallel data, as the alignment reaches values of 0.9 in average.

Llama 3.1 8B achieves better MEXA alignment scores across all pairs compared to CodeLlama 7B. This is not entirely unexpected: even though CodeLlama 7B and its instruction-tuned version are specifically trained for code, newer generic models of Llama, including Llama 3 8B (Dubey et al., 2024) and its instruction-tuned version, achieve better scores in code generation tasks (as evaluated on LiveCodeBench (Jain et al., 2024)).[1]

3.3 Method 3: Language-specific neurons

We identify language-specific neurons for 11 PLs and English using LAPE. PPL change results ($\nu = 400, \tau = 0.95$) in Figure 4 show that deactivating language-specific neurons has negligible effects on other languages, while more noticeably impacting the primary language—though this may not hold across all settings. For other ν values, we apply the LAPE neuron identification method and measure PPL changes by incrementally deactivating language-specific neurons for each primary language. The results for $\tau = 0.95$ are shown in

[1] hf.co/spaces/livecodebench/leaderboard



(a) Code Llama 7B

(b) Llama 3.1 8B

Figure 5: Impact of LAPE neuron identification. X-axis: Number of shared neurons for each language. Y-axis: Change in PPL across languages when deactivating the primary language's neurons (e.g., English in the lower-right figure). Figure 7 in Appendix A shows the same figure for more languages.

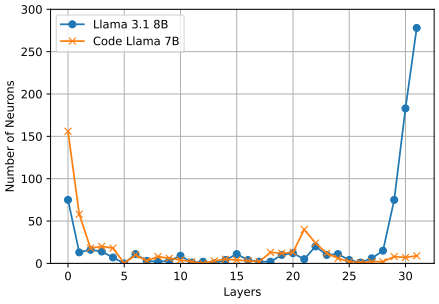


Figure 6: Number of "language-specific" neurons, i.e., neurons that are exclusive to one PL and not shared with any other PL, in the LAPE experiment with $\nu = 1400, \tau = 0.95$. The figure shows the total number of language-specific neurons summed over all PLs.

Figure 5, where we observe that LAPE fails to identify "effective" language-specific neurons for some languages. Effectiveness is indicated by a larger PPL change for the primary language compared to other languages when deactivating the primary language-specific neurons.

In most cases, increasing the number of neurons enlarges the PPL gap between primary and other languages in Llama 3.1. However, for C# and Java (and C++ and HTML in Figure 7) the gap is less pronounced. Interestingly, C# and Java are the PLs with the highest alignment in Llama 3.1, as shown in Section 3.2. Additionally, C# has one of the largest PL keyword sets appearing frequently in intermediate layers, as shown in Section 3.1. This suggests that the identified specific neurons for these languages are more shared across languages. In other words, for PLs such as C# and Java, which closely align with multiple other PLs, distinguishing language-specific neurons is more challenging.

For CodeLlama 7B, even though some effective language-specific neurons exist, the PPL change is not significant (Figures 4a, 5a). When the number of deactivated neurons exceeds 12,500, the impact on other languages sometimes surpasses that on the primary language. The only language for which CodeLlama 7B identifies effective specific neurons with a larger PPL change margin is English. This suggests that PL neurons in CodeLlama 7B are highly shared, possibly due to its training recipe, where the pre-training phase following Llama 2 initialization primarily focuses on code.

To further investigate what makes language-specific neurons effective in Llama 3.1 8B but not in CodeLlama 7B, we examine the language-specific neurons selected for all PLs that are "exclusive" to each PL, as shown in Figure 6 for $\nu = 1400, \tau = 0.95$. Other ν settings exhibit a similar distribution. In general, most language-specific neurons in both models and across most languages are selected from the bottom layers (indices 0 to 4), followed by layer indices 18 to 22 in both models. However, those that are exclusive to a specific PL are predominantly selected from the top layers (indices 29 to 31). Notably, LAPE selects more exclusive neurons from top layers for Llama 3.1 8B than CodeLlama 7B as shown in Figure 6. This aligns with the fact that top layers serve for token generation, where the LLM must handle the

"output space" and map it to the expected token. If exclusive language-specific neurons exist for each primary language at the top layers, deactivating that language's neurons only affects the PPL of that language. However, if there are no such exclusive neurons, it affects PPL of other languages as well.

4 Discussion and Implications

Our findings suggest several strategies for building more efficient multilingual code models.

1) Since English and certain PLs are centrally located in the model's concept space, these could serve as intermediate representations for multilingual code translation, minimizing the distance between source and target languages.

2) The distribution of neuron types across layers – shared/general in bottom layers, specific in top – supports modular architectures where base layers encode general syntax/semantics and top layers can be swapped or specialized for specific languages.

3) For closely aligned PLs (e.g., Java and C#), shared representations could enable parameter sharing or adapter-based methods for lightweight multilingual support, while only tuning minimal additional weights.

4) Some languages enforce object-oriented programming, while others support it optionally. This structural difference may lead the model to develop stronger internal representations for languages with stricter paradigms, potentially introducing some bias in code generation. Other differences in language design and idiomatic usage can influence the model's behavior when generating code across languages. Recognizing these factors could help improve the generalization capabilities of multilingual code models.

5 Related work

Pivot language. Wendler et al. (2024) use logit lens (Nostalgebraist, 2020) to show that English acts as a kind of "pivot" language in English-centric LLMs, such as Llama-2 (Touvron et al., 2023b), enabling these models to solve complex semantic tasks in a non-English language by detouring through English internal states before generating non-English text. Building on this idea, Wu et al. (2025) propose the semantic hub hypothesis, which suggests that the same phenomenon could occur not only across different languages but also across different modalities. As one of these modalities, they introduce code. Their analysis focuses solely on Python within the Llama 2 model. Since obtaining semantically equivalent English-Python pairs is challenging, they test only a few targeted cases, such as the English token "and" and its Python counterpart ",". Using logit lens, they show that in the intermediate layers, the expected Python token is closer to "and" than to other tokens such as "or" and "not." In our work, we focus exclusively on PLs and consider seven PLs. As noted by Wu et al. (2025), obtaining semantically equivalent English-PL pairs is challenging. Instead, we analyze keyword sets—comprising keywords from 22 PLs and an English dictionary—through a translation task across 42 directions. This allows us to examine which PLs and English-derived tokens appear in the model's intermediate layers and are closer to its concept space, both in terms of probability and rank. Our findings reveal that not only English but also other PLs contribute to the model's concept space.

Neuron-level interpretability. Initially, language-specific components were studied in neural machine translation using small language models (Lin et al., 2021; Xie et al., 2021; Zhang et al., 2021). Later, the role of FFNs within LLMs was explored in several studies, highlighting their function as key-value memories for storing factual and linguistic knowledge (Geva et al., 2021, 2022; Ferrando et al., 2023). However, these analyses typically investigate neuron behavior, focusing on monolingual settings in natural languages and PLs. Building on methods explored in investigations on the role of FFNs within LLMs and considering clear evidence that LLMs exhibit significant overlap in their embeddings across languages—particularly among those from the same linguistic family (Doddapaneni et al., 2021)—several recent studies (Xie et al., 2021; Tang et al., 2024; Zhao et al., 2024; Kojima et al., 2024; Wang et al., 2024; Bhattacharya and Bojar, 2023, 2024; Mueller et al., 2022; Liu et al., 2024a; Dumas et al., 2024; Liu et al., 2025) have investigated the existence of language-specific neurons and internal mechanisms for natural languages, especially within the FFN layers of LLMs. Just as there are many natural languages, there are also many PLs. However, no research has explored the existence of language-specific neurons for PLs, even though LLMs are typically pre-trained on a mixture of these languages. Building on this, our work adopts the method proposed by Tang

et al. (2024) to identify PL-specific neurons. This approach enables a scalable and targeted analysis of neurons for many PLs using only raw PL data.

Interpretability for code. Interpretability in language models for code-related tasks remains underexplored, with most research focusing on attention layers (Mohammadkhani et al., 2023; Wan et al., 2022; Paltenghi and Pradel, 2021; Liu et al., 2024b). Our work is closest to Haider et al. (2024), who analyze FFN layers. They analyze two GPT-based models (Xu et al., 2022; Nijkamp et al., 2022) for three PLs, showing that lower layers capture syntax while higher layers encode abstract concepts and semantics. They demonstrate that concepts are stored in the FFN layers and can be edited without compromising code language model performance. However, their analysis is performed in monolingual settings, while our work investigates the relationship between PLs to determine if they share concepts and neurons in coding tasks.

6 Conclusion

In this study, we investigate how LLMs represent programming languages (PLs) in their concept space using the logit lens method. We observe that English and PL keywords appear in intermediate layers, with notable probabilities in the latter half. Initially, these keywords surpass the expected output tokens, but as the probabilities of expected tokens increase and overtake those of English and PL keywords, the probabilities of the latter decline. We further investigate the existence of language-specific neurons using the language activation probability entropy (LAPE) method. Our analysis reveals that language-specific neurons can be identified for most languages in the Llama 3.1 model, but not for PLs such as Java and C#, which align closely with other PLs. We find that language-specific neurons are concentrated in the bottom layers, while neurons exclusive to each PL are located in the top layers. These findings deepen our understanding of LLMs' inner workings in the context of PLs and provide valuable insights for interpretability in code-related tasks.

Limitations

We are aware of three main limitations of our work.

First, parts of our analysis rely on a super-parallel dataset, which is limited to seven languages due to source constraints. To our knowledge, no super-parallel dataset with a broader language set is publicly available. A potential solution is to generate super-parallel data for more languages using more powerful LLMs and validate it through unit tests to ensure quality and consistency.

Second, while we use keywords to interpret latent embeddings, a more precise approach would involve constructing a dictionary mapping PL keywords to each other and their English equivalents. However, this is not always feasible, as some PL keywords lack direct English meanings or map to multiple tokens.

Third, we hypothesize that the ineffectiveness of neuron identification for CodeLlama 7B stems from its training recipe, but further investigation across other models could be beneficial. Our analysis focuses on PLs in Llama-based architectures, which underlie many state-of-the-art models, but it's important to explore other architectures for broader validation.

Acknowlegments

This research was supported by DFG (grant SCHU 2246/14-1). François Yvon has been partly funded by the French National Funding Agency (ANR) under the France 2030 program (ref. ANR-23-IACL-0007).

References

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. 2024. To code, or not to code? exploring impact of code in pre-training. *Preprint*, arXiv:2408.10914.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens. *Preprint*, arXiv:2303.08112.

Sunit Bhattacharya and Ondřej Bojar. 2023. Unveiling multilinguality in transformer models: Exploring language specificity in feed-forward networks. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 120–126, Singapore. Association for Computational Linguistics.

Sunit Bhattacharya and Ondřej Bojar. 2024. Understanding the role of ffns in driving multilingual behaviour in llms. *Preprint*, arXiv:2404.13855.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large

language models trained on code. *arXiv preprint arXiv:2107.03374*.

CodeParrot. 2022. GitHub code dataset.

DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu, Yukun Li, Huazuo Gao, Shirong Ma, and 1 others. 2024. DeepSeek-Coder-v2: Breaking the barrier of closed-source models in code intelligence. *Preprint*, arXiv:2406.11931.

Sumanth Doddapaneni, Gowtham Ramesh, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. 2021. A primer on pretrained multilingual language models. *Preprint*, arXiv:2107.00676.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Clément Dumas, Veniamin Veselovsky, Giovanni Monea, Robert West, and Chris Wendler. 2024. How do llamas process multilingual text? a latent exploration through activation patching. In *ICML 2024 Workshop on Mechanistic Interpretability*.

Javier Ferrando, Gerard I. Gállego, Ioannis Tsiamas, and Marta R. Costa-jussà. 2023. Explaining how transformers use context to build predictions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5486–5513, Toronto, Canada. Association for Computational Linguistics.

GeeksforGeeks. 2008. Geeksforgeeks: A computer science portal for geeks.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

GitHut. 2024. GitHut 2.0: Language popularity in GitHub repositories.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Muhammad Umair Haider, Umar Farooq, A. B. Siddique, and Mark Marron. 2024. Looking into black box code language models. *Preprint*, arXiv:2407.04868.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *Preprint*, arXiv:2406.00515.

David Kamholz, Jonathan Pool, and Susan Colowick. 2014. PanLex: Building a resource for panlingual lexical translation. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 3145–3150, Reykjavik, Iceland. European Language Resources Association (ELRA).

Amir Hossein Kargaran, Ali Modarressi, Nafiseh Nikeghbal, Jana Diesner, François Yvon, and Hinrich Schütze. 2024. MEXA: Multilingual evaluation of english-centric LLMs via cross-lingual alignment. *Preprint*, arXiv:2410.05873.

Takeshi Kojima, Itsuki Okimura, Yusuke Iwasawa, Hitomi Yanaka, and Yutaka Matsuo. 2024. On the multilingual ability of decoder-based pre-trained language models: Finding and controlling language-specific neurons. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6919–6971, Mexico City, Mexico. Association for Computational Linguistics.

Marie-Anne Lachaux, Baptiste Rozière, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Preprint*, arXiv:2006.03511.

Raymond Li, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, LI Jia, Jenny Chim, Qian Liu, and 1 others. 2023. Starcoder: may the source be with you! *Transactions on Machine Learning Research*.

Zehui Lin, Liwei Wu, Mingxuan Wang, and Lei Li. 2021. Learning language specific sub-network for multilingual machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*

(Volume 1: Long Papers), pages 293–305, Online. Association for Computational Linguistics.

Weize Liu, Yinlong Xu, Hongxia Xu, Jintai Chen, Xuming Hu, and Jian Wu. 2024a. Unraveling babel: Exploring multilingual activation patterns of LLMs and their applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11855–11881, Miami, Florida, USA. Association for Computational Linguistics.

Yihong Liu, Runsheng Chen, Lea Hirlimann, Ahmad Dawar Hakimi, Mingyang Wang, Amir Hossein Kargaran, Sascha Rothe, François Yvon, and Hinrich Schütze. 2025. On relation-specific neurons in large language models. *Preprint*, arXiv:2502.17355.

Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, and Li Li. 2024b. On the reliability and explainability of language models for program generation. *ACM Transactions on Software Engineering and Methodology*, 33(5):1–26.

Michael R Lyu, Baishakhi Ray, Abhik Roychoudhury, Shin Hwei Tan, and Patanamon Thongtanunam. 2024. Automatic programming: Large language models and beyond. *ACM Transactions on Software Engineering and Methodology*.

Nick Meyer and Leigh McCulloch. 2022. Keywords: A list and count of keywords in programming languages. https://github.com/e3b0c442/keywords.

Ahmad Haji Mohammadkhani, Chakkrit Tantithamthavorn, and Hadi Hemmatif. 2023. Explaining transformer-based code models: What do they learn? when they do not work? In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 96–106. IEEE.

Aaron Mueller, Yu Xia, and Tal Linzen. 2022. Causal analysis of syntactic agreement neurons in multilingual language models. In *Proceedings of the 26th Conference on Computational Natural Language Learning (CoNLL)*, pages 95–109, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Niklas Muennighoff. 2022. Sgpt: Gpt sentence embeddings for semantic search. *Preprint*, arXiv:2202.08904.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Nostalgebraist. 2020. Interpreting GPT: The logit lens. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Matteo Paltenghi and Michael Pradel. 2021. Thinking like a developer? comparing the attention of humans with neural models of code. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 867–879. IEEE.

Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. 2025. CodeElo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *Preprint*, arXiv:2501.01257.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Robert W. Sebesta. 2016. *Concepts of Programming Languages*, 11 edition. Pearson Education Limited, Harlow, England.

Noam Shazeer. 2020. Glu variants improve transformer. *Preprint*, arXiv:2002.05202.

Tianyi Tang, Wenyang Luo, Haoyang Huang, Dongdong Zhang, Xiaolei Wang, Xin Zhao, Furu Wei, and Ji-Rong Wen. 2024. Language-specific neurons: The key to multilingual capabilities in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5701–5715, Bangkok, Thailand. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Yao Wan, Wei Zhao, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2022. What do they capture? a structural analysis of pre-trained language models for source code. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2377–2388.

Weixuan Wang, Barry Haddow, Minghao Wu, Wei Peng, and Alexandra Birch. 2024. Sharing matters: Analysing neurons across languages and tasks in llms. *Preprint*, arXiv:2406.09265.

Chris Wendler, Veniamin Veselovsky, Giovanni Monea, and Robert West. 2024. Do llamas work in english? on the latent language of multilingual transformers. *Preprint*, arXiv:2402.10588.

Zhaofeng Wu, Xinyan Velocity Yu, Dani Yogatama, Jiasen Lu, and Yoon Kim. 2025. The semantic hub hypothesis: Language models share semantic representations across languages and modalities. In *International Conference on Learning Representations (ICLR)*.

Wanying Xie, Yang Feng, Shuhao Gu, and Dong Yu. 2021. Importance-based neuron allocation for multilingual neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5725–5737, Online. Association for Computational Linguistics.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10.

Biao Zhang, Ankur Bapna, Rico Sennrich, and Orhan Firat. 2021. Share or not? learning to schedule language-specific capacity for multilingual translation. In *Ninth International Conference on Learning Representations 2021*.

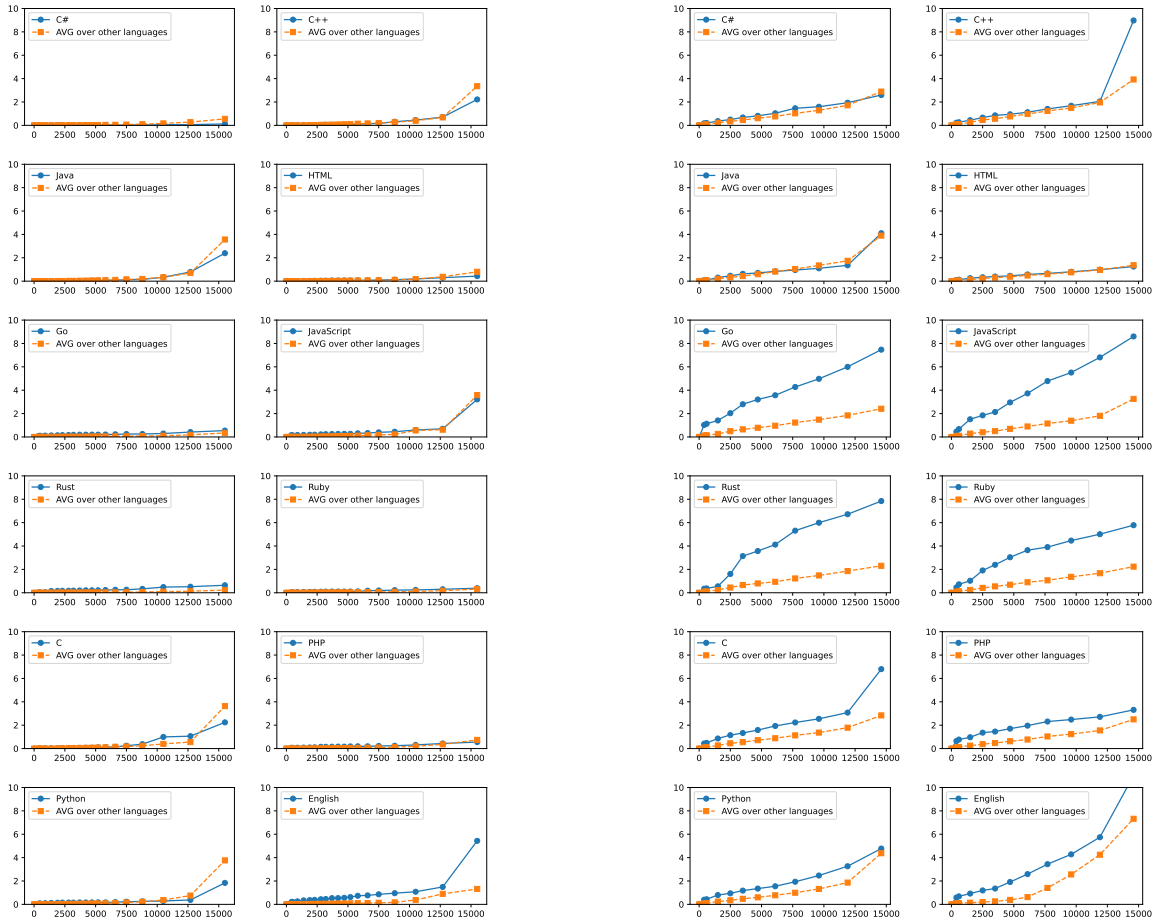
Yiran Zhao, Wenxuan Zhang, Guizhen Chen, Kenji Kawaguchi, and Lidong Bing. 2024. How do large language models handle multilingualism? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Ming Zhu, Aneesh Jain, Karthik Suresh, Roshan Ravindran, Sindhu Tipirneni, and Chandan K. Reddy. 2022a. Xlcost: A benchmark dataset for cross-lingual code intelligence. *Preprint*, arXiv:2206.08474.

Ming Zhu, Karthik Suresh, and Chandan K Reddy. 2022b. Multilingual code snippets training for program translation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 11783–11790.

A Impact of LAPE neuron identification

We show the complete version of Figure 5 in Figure 7, covering more PLs.



(a) Code Llama 7B

(b) Llama 3.1 8B

Figure 7: Impact of LAPE neuron identification. X-axis: Number of shared neurons for each language. Y-axis: Change in PPL across languages when deactivating the primary language's neurons (e.g., English in the lower-right figure).