



The Rise and Down of Babel Tower: Investigating the Evolution Process of Multilingual Code Large Language Model

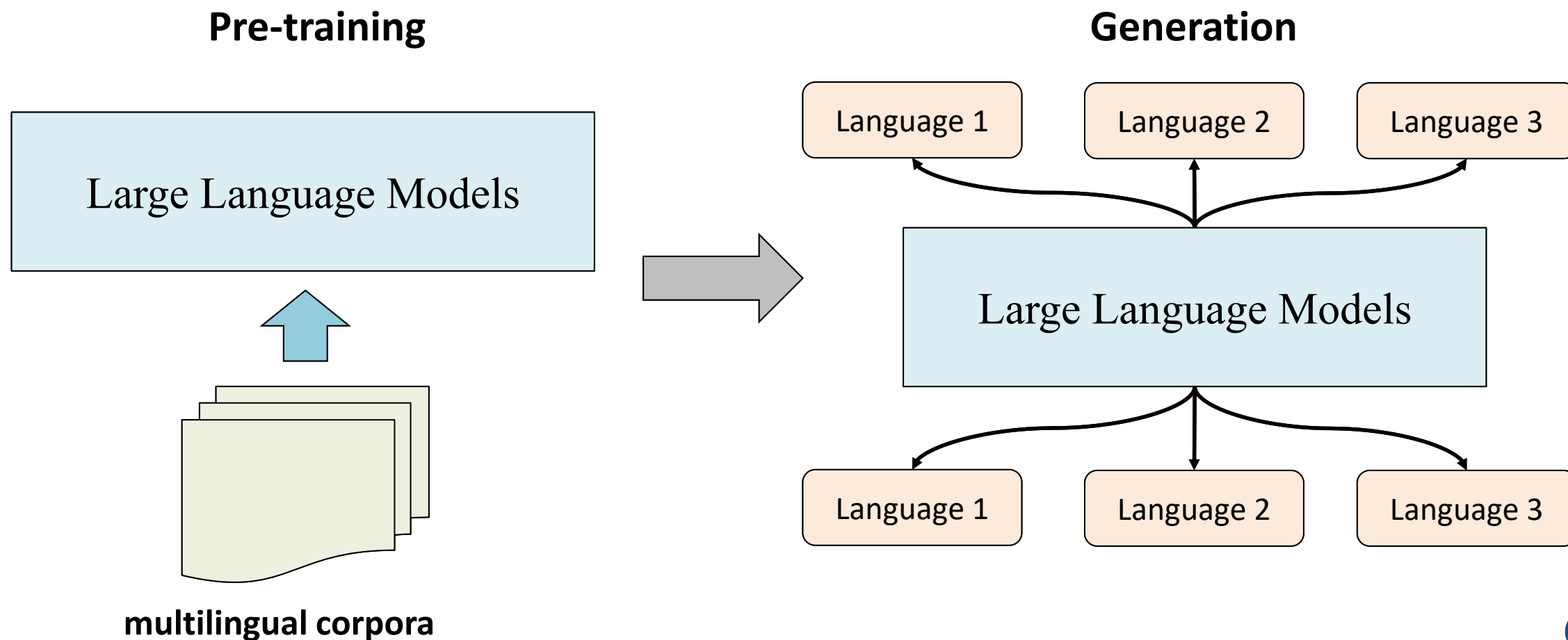
**Jiawei Chen, Wentao Chen, Jing Su, Jingjing Xu, Hongyu Lin,
Mengjie Ren, Yaojie Lu, Xianpei Han, Le Sun**

**Chinese Information Processing Laboratory, Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
ByteDance Inc.**

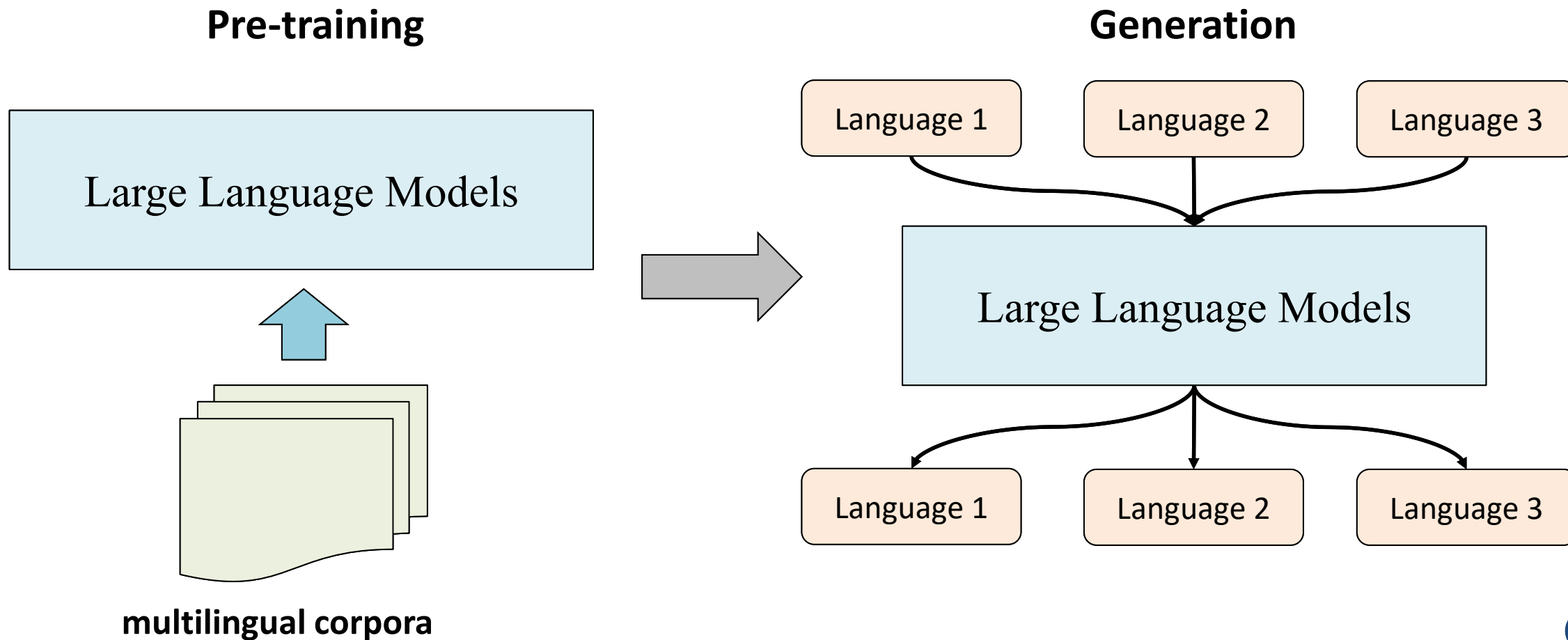
- Introduction
- The Evolution Process of Multilingual Systems
- Achieving Optimal Multilingual Performance
- Conclusions

- Introduction
- The Evolution Process of Multilingual Systems
- Achieving Optimal Multilingual Performance
- Conclusions

Large Language Models: Multilingual Capabilities

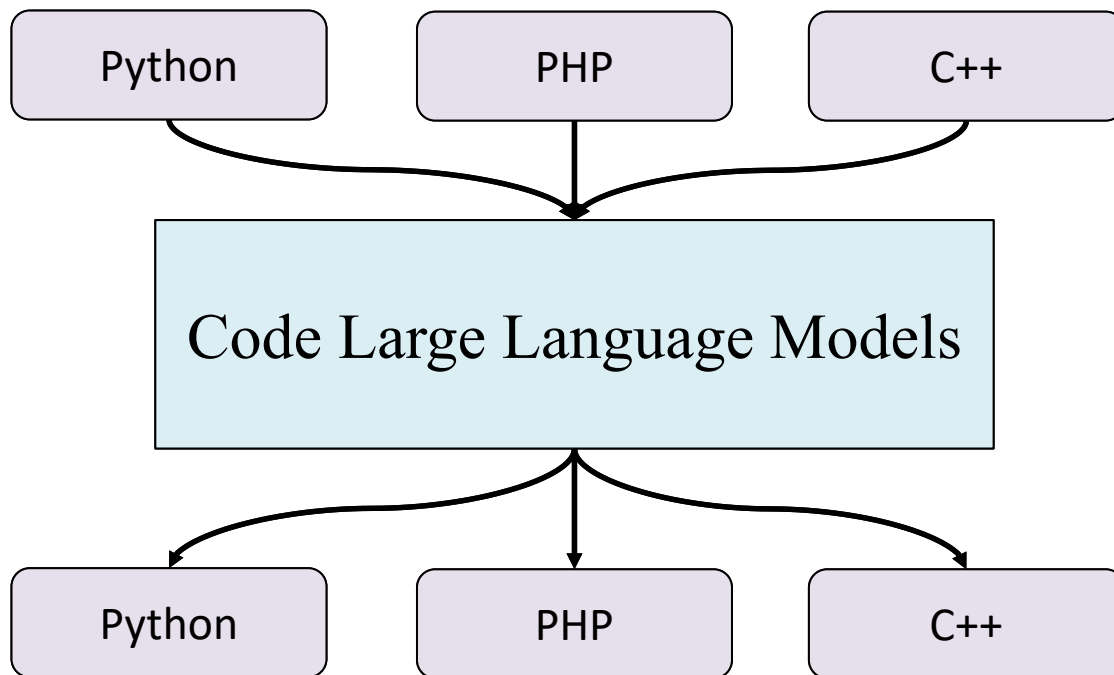


How do the multilingual capabilities of LLMs evolve during pre-training process?



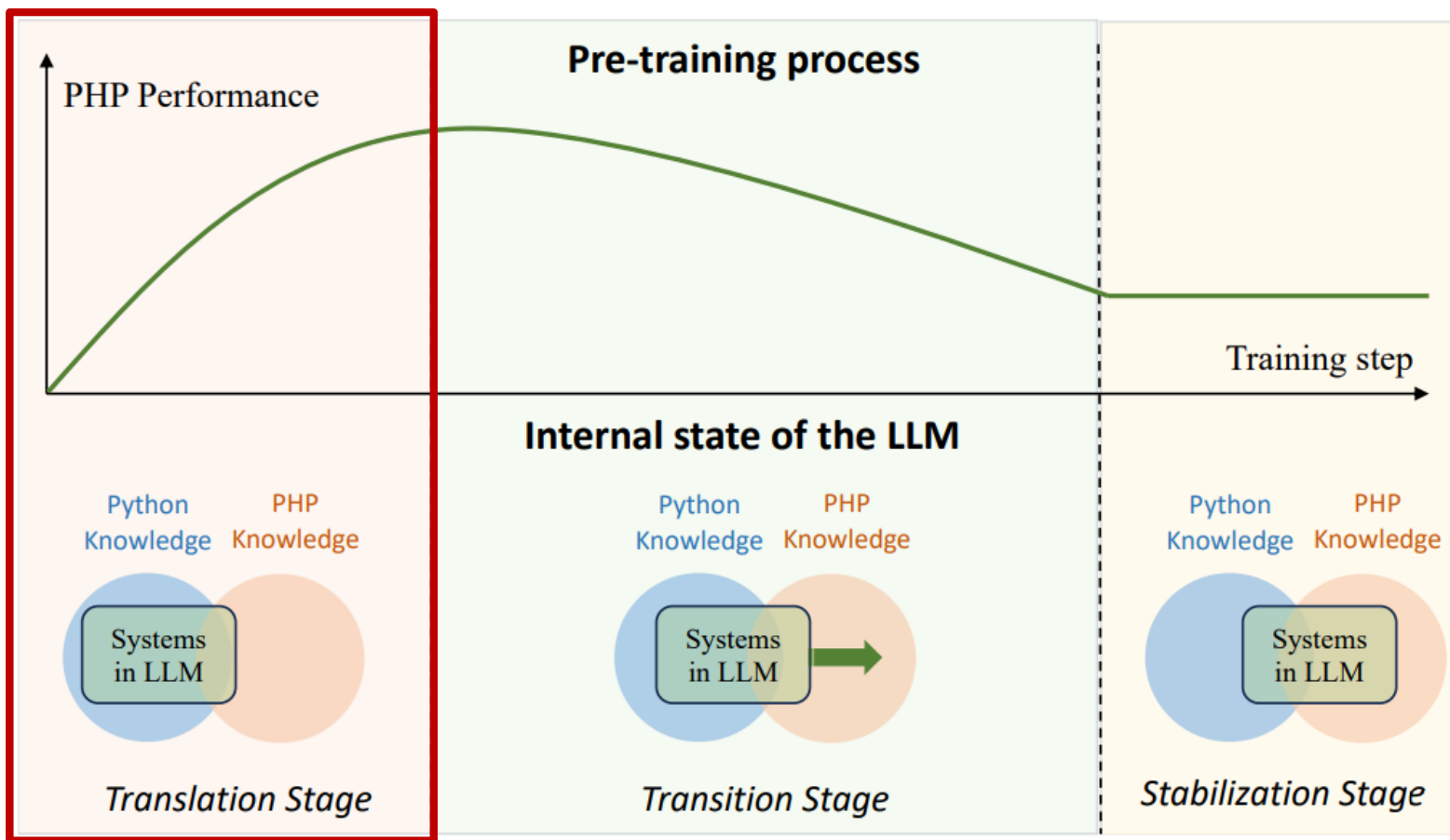
- We use **code LLMs** as an experimental platform

Generating programming languages

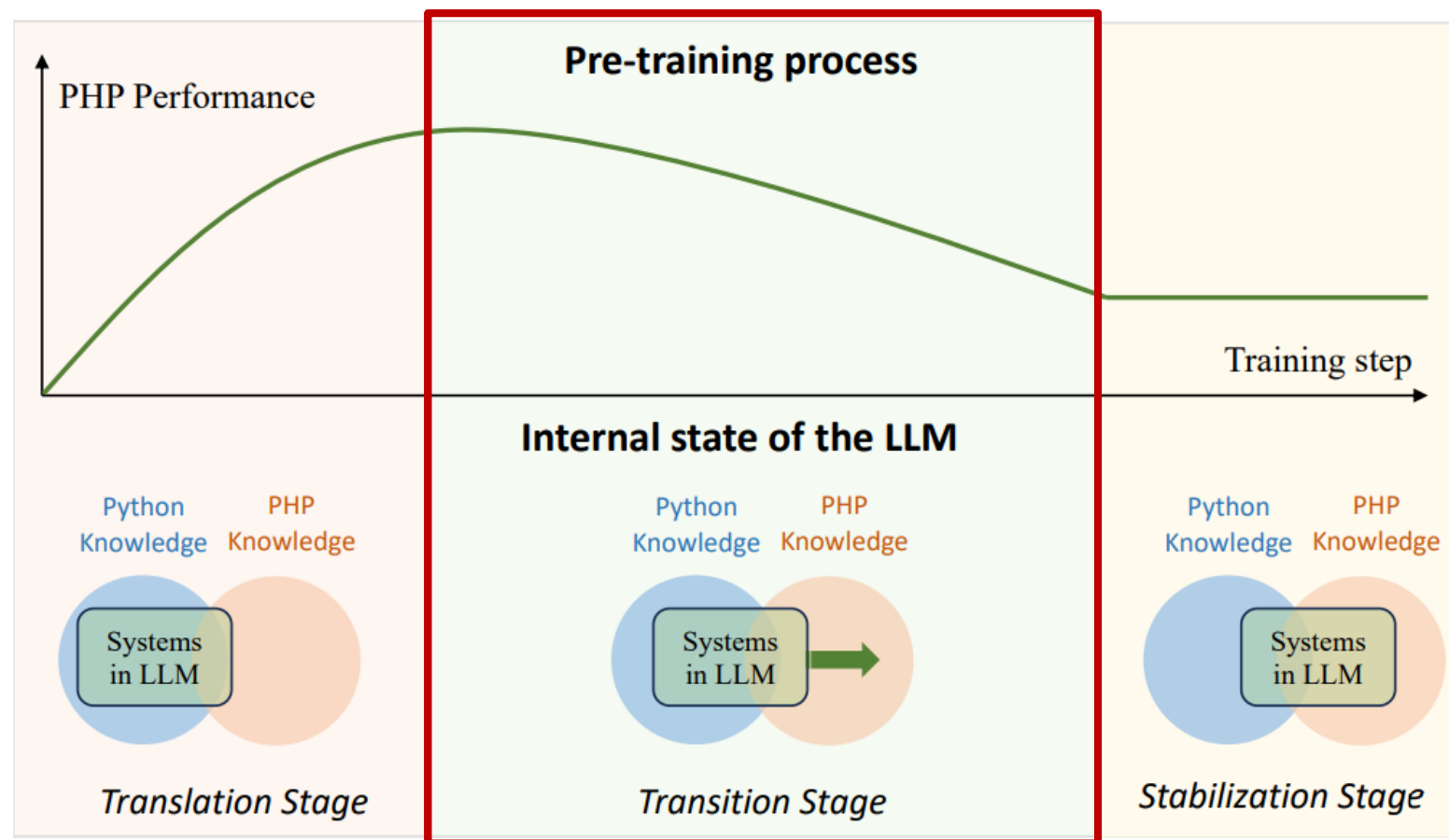


- Preliminary experiments: how monolingual LLMs learn anew language

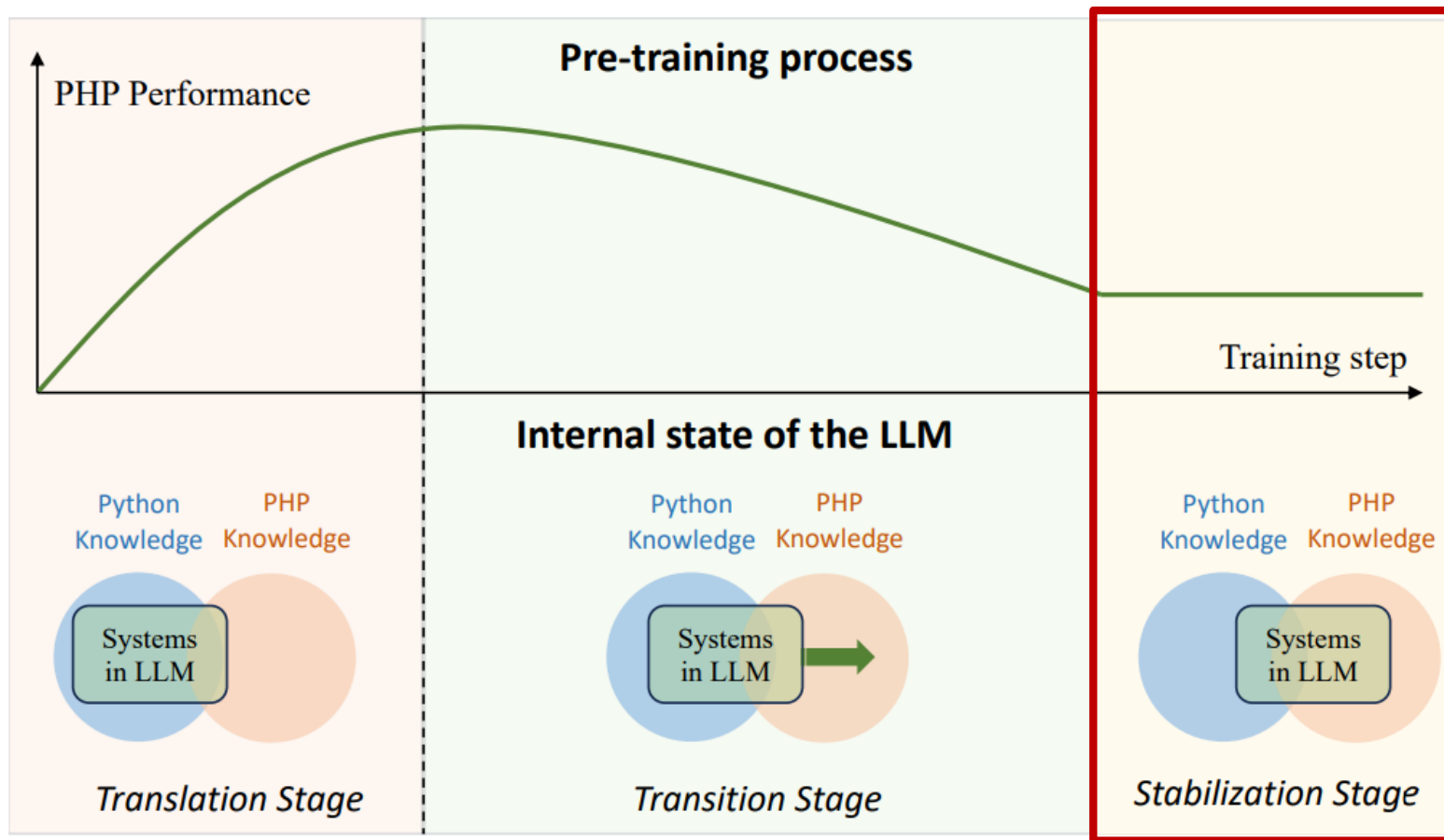
- Preliminary experiments: how monolingual LLMs learn anew language



- Preliminary experiments: how monolingual LLMs learn anew language



- Preliminary experiments: how monolingual LLMs learn anew language



The Evolution Process of Multilingual LLMs



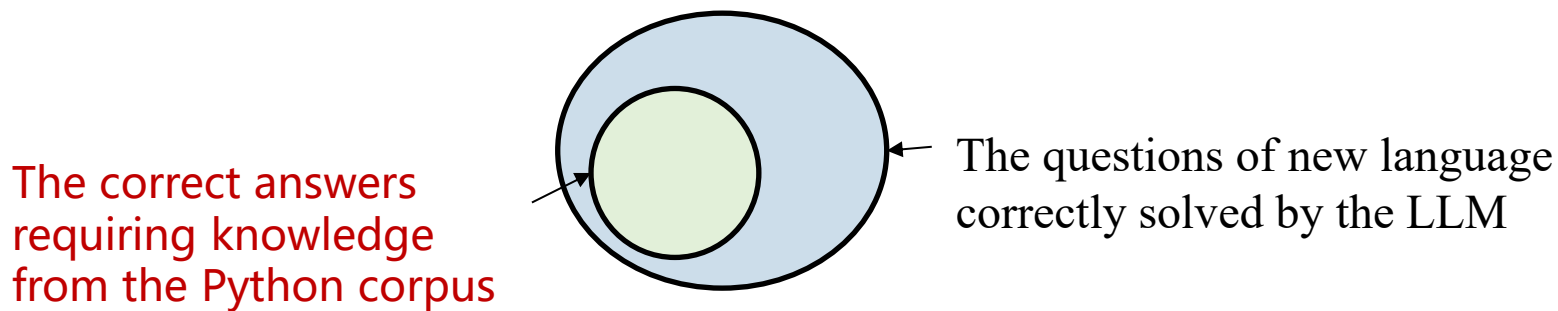
multiple languages initially share a **single** knowledge system dominated by a primary language
and then gradually shift to developing **multiple** language-specific knowledge systems as training in new languages progresses

- Introduction
- The Evolution Process of Multilingual Systems
- Achieving Optimal Multilingual Performance
- Conclusions

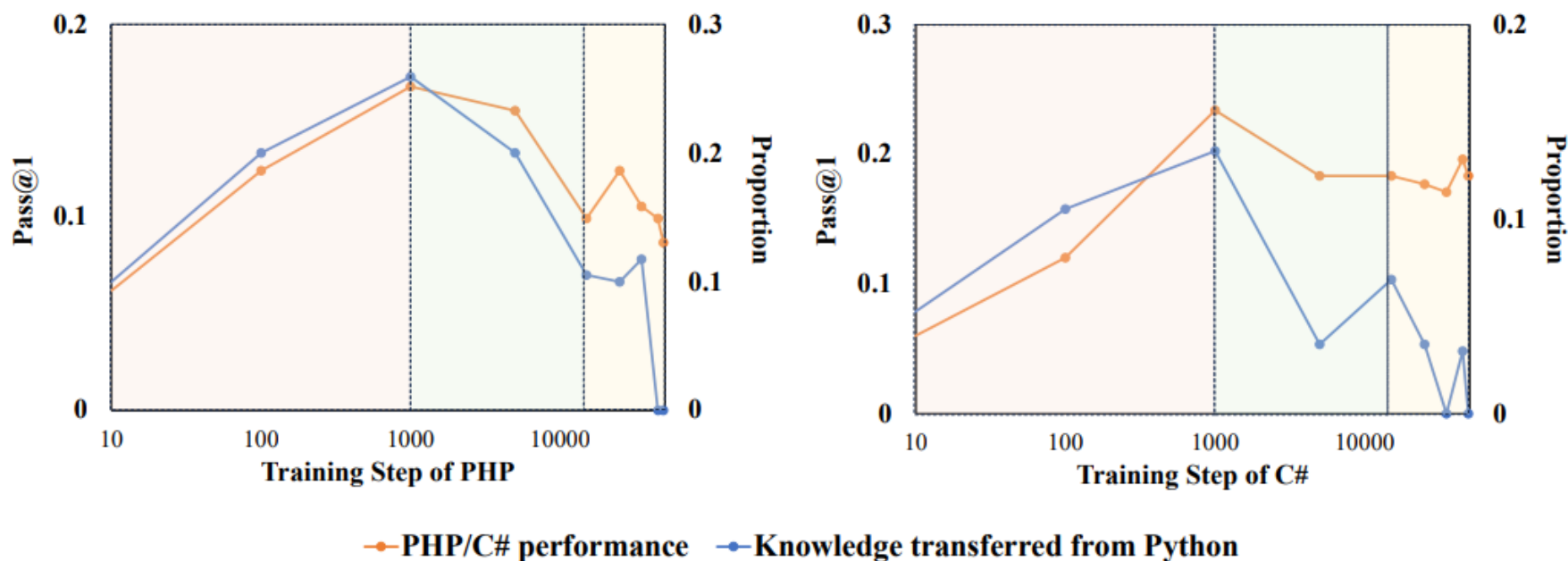
- Experimental Setup
 - Dominate Language: Python
 - Main experiment: **Learning a new language**
 - Pre-training the monolingual Python LLM using corpus other languages such as PHP, C#, Go, or C++
 - We analyze the performance of **new language** in **HumanEval**.

The sources of the capabilities of the LLM during pre-training

- **Metric:** *The proportion of correct answers requiring knowledge from the Python corpus*



The sources of the capabilities of the LLM during pre-training

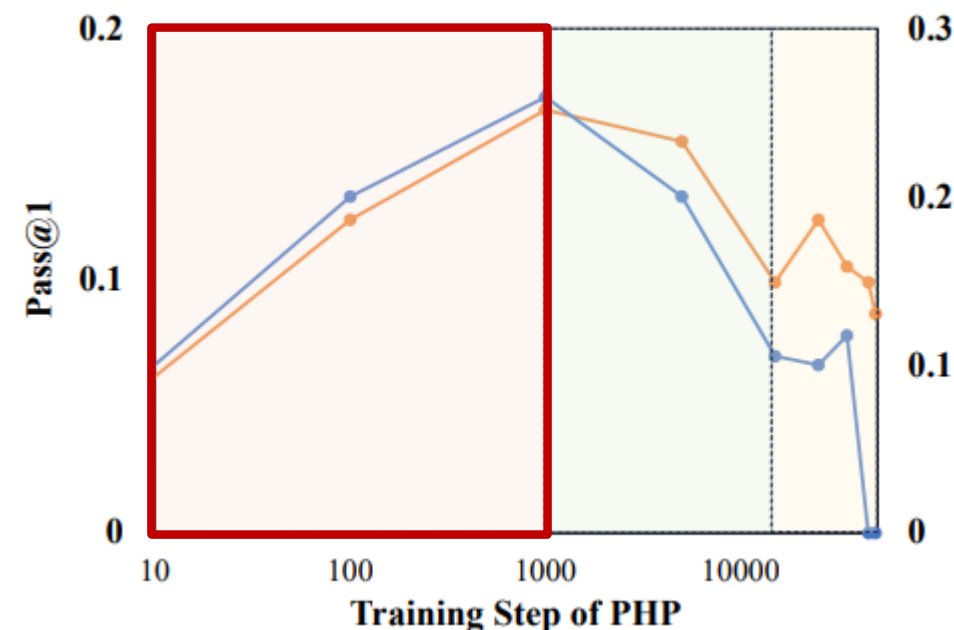


The pre-training process can be divided into three stages

The sources of the capabilities of the LLM during pre-training

- **Translation stage**

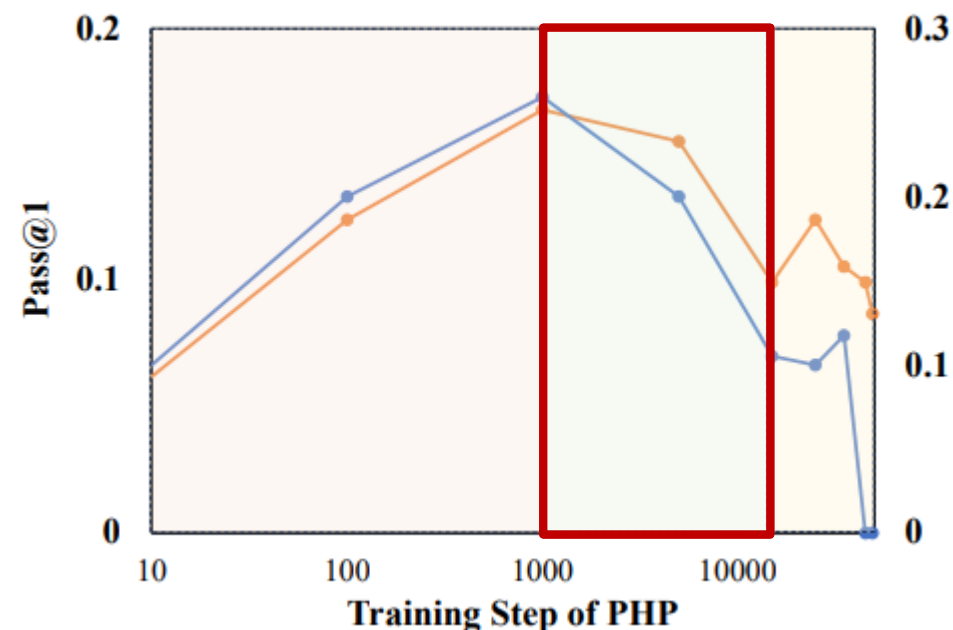
- In the initial stage of pre-training, the performance of PHP and C# improved significantly.
- Concurrently, the proportion of correct answers necessarily relying on Python knowledge was notably high.
- We hypothesize that the LLMs are utilizing their Python knowledge to generate solutions for PHP and C# problems.



The sources of the capabilities of the LLM during pre-training

- **Transition stage**

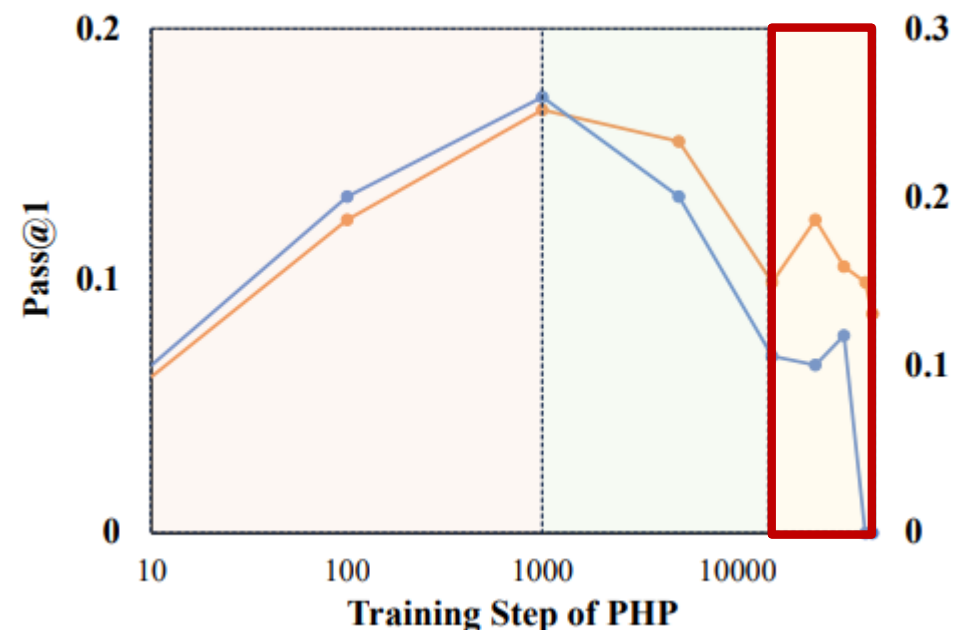
- Subsequently, the performance gradually declined, accompanied by a decrease in the proportion of correct answers necessarily relying on Python knowledge.
- This could be because the PHP/C# system gradually formed, leading to a gradual transformation from the Python system to the PHP/C# system during generation.



The sources of the capabilities of the LLM during pre-training

- **Stabilization stage**

- Finally, the performance stabilized, there are almost no correct answers that necessarily require Python knowledge to solve.
- We hypothesize that the LLM might have reached a stable internal state dominated by PHP/C#.



Babel Tower Hypothesis

During the learning process of large language models (LLMs),
multiple languages initially share a single knowledge system dominated by a primary language
and then gradually shift to developing multiple language-specific knowledge systems as training in new languages progresses.

Validation1: Working Languages

We employ the logit to generate predicted tokens in **intermediate layers**, which can be seen as the **working languages** of the LLMs.

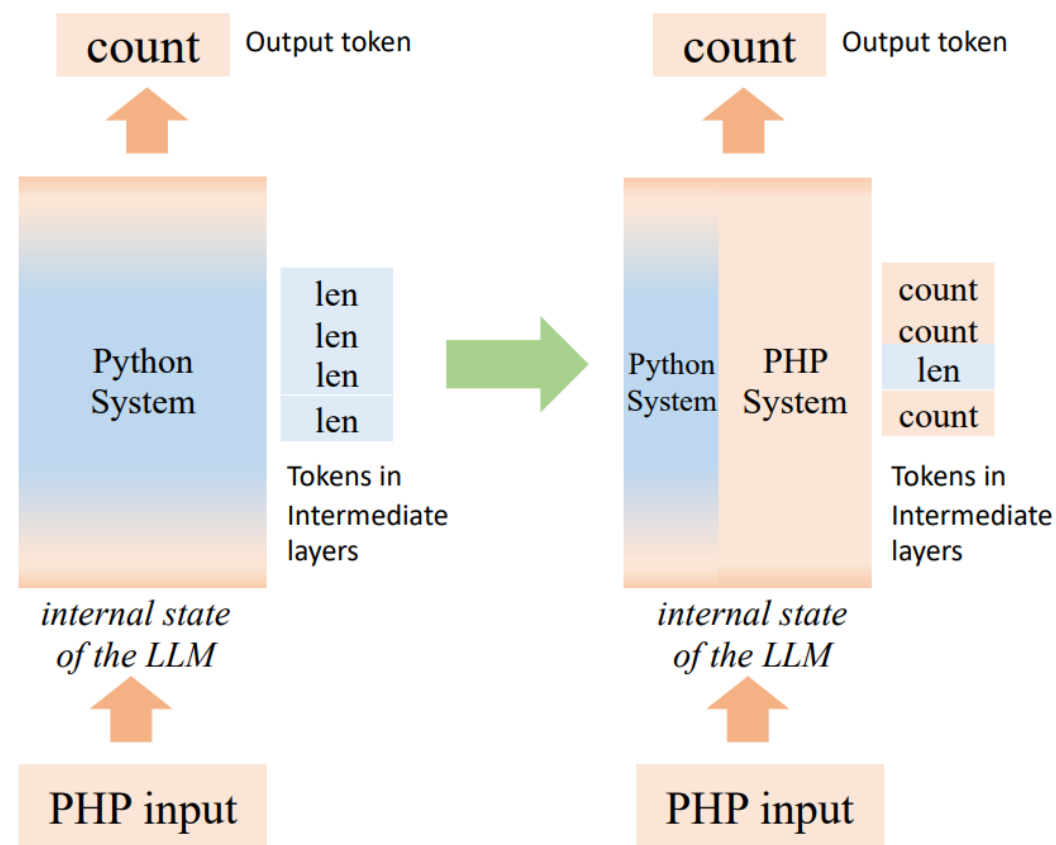
We count the number of generated tokens in each language and calculate the frequency of for each language.

This frequency is used to estimate the proportion of working language.

Validation1: Working Languages

To accurately determine the working language, we use **built-in functions or statements** with identical functionalities but different tokens as identifiers.

For example, *to return the length of an array*, Python uses the “*len()*” function, while PHP uses the “*count()*” function.

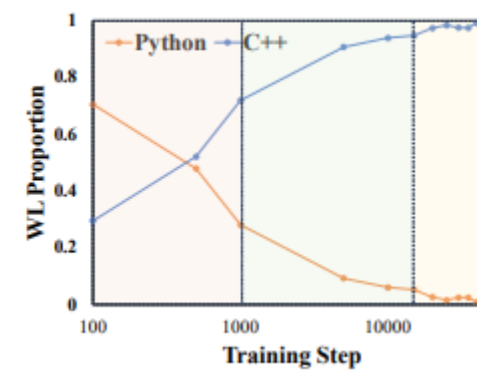
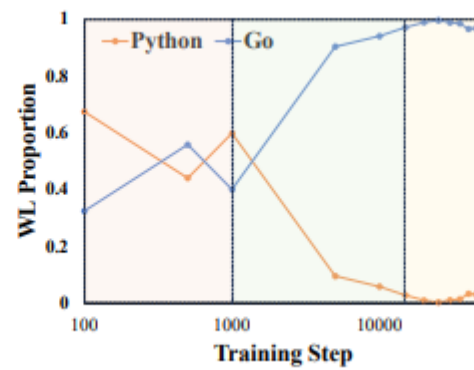
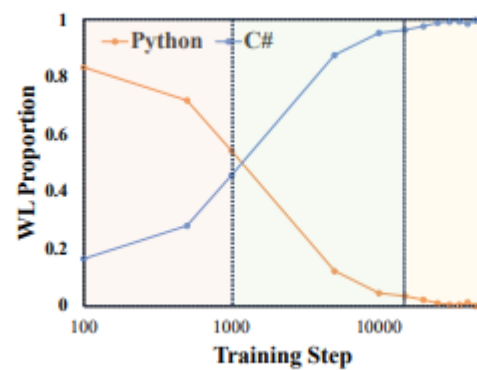
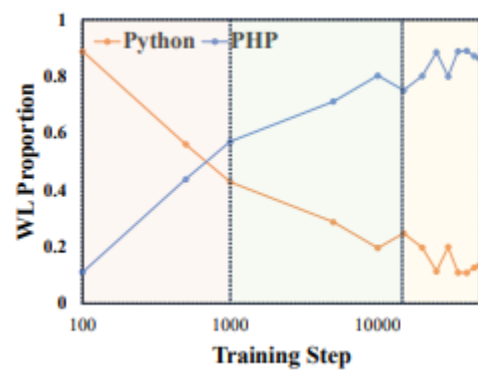


Validation2: Language transferring Neurons

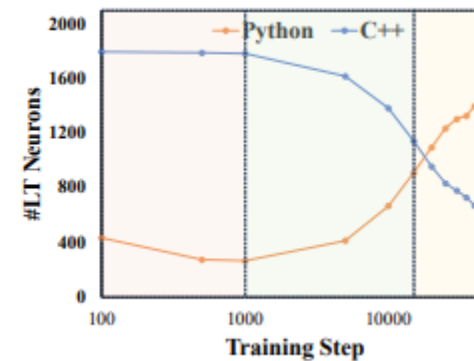
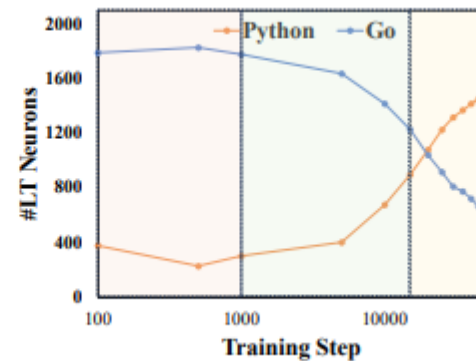
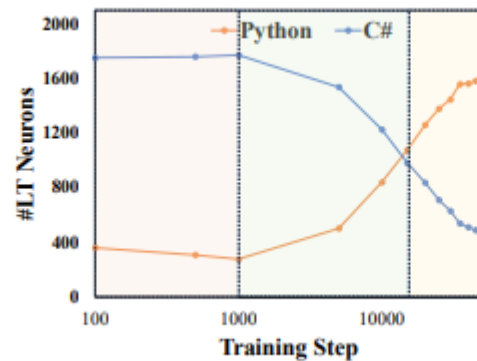
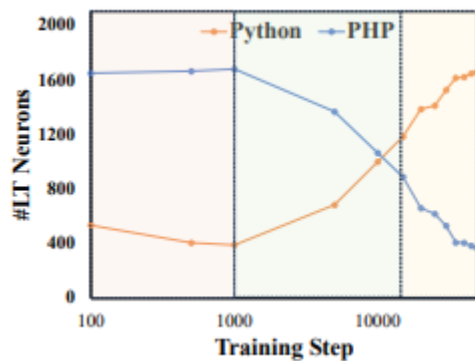
There are language-specific neurons which will be activated in response to particular languages, and they are typically distributed across both the **bottom and top** layers in the LLM representing **the transformation process** from input to intermediate layers and from intermediate layers to output.

The number of language-transferring neurons per language serves as the final metric. When the primary language system dominates the LLM, a large number of neurons will be involved in language transfer for other languages.

Validation Result



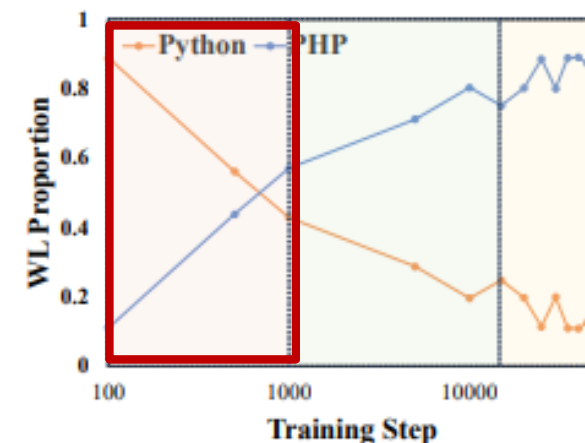
The proportion of a language being the working language.



The number of language transferring neurons in different languages

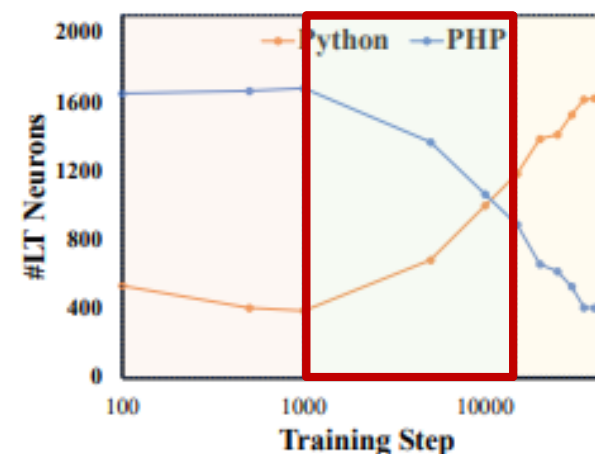
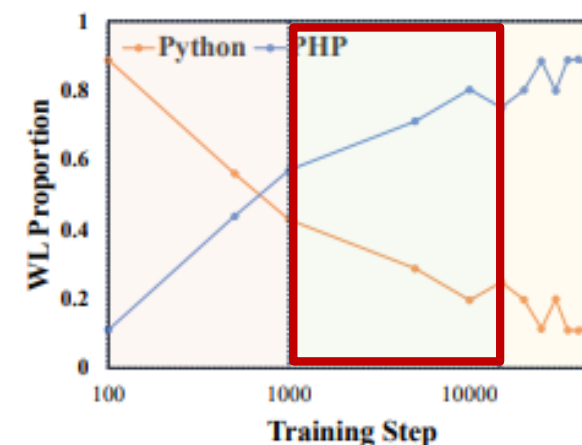
Validation Result

- **Translation stage**
 - Python serves as the primary working language.
 - The number of language transferring neurons activated for the new language is significantly higher .
 - This indicates that the LLM activates extra neurons to transform Python knowledge into tokens of the new language, akin to a translation process.



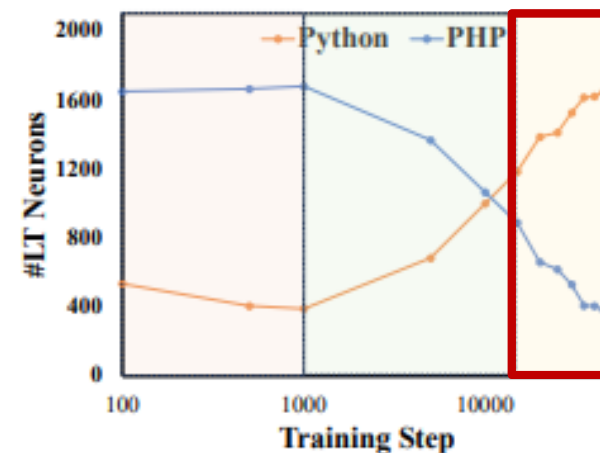
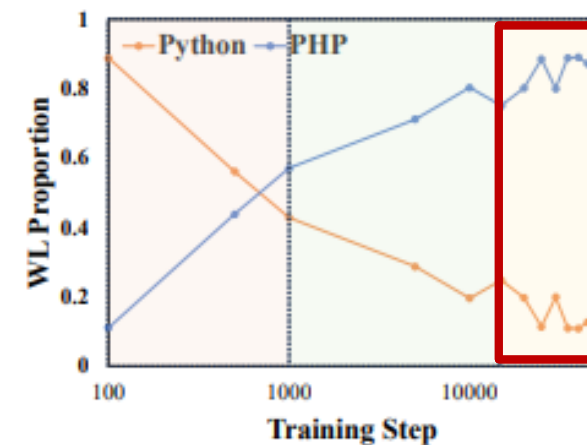
Validation Result

- **Transition stage**
 - The working language gradually transitions from Python to the new language
 - The proportion of neurons related to new languages decreases accordingly.
 - This indicates that the Python system gradually shifts to the new language system during the entire pre-training process



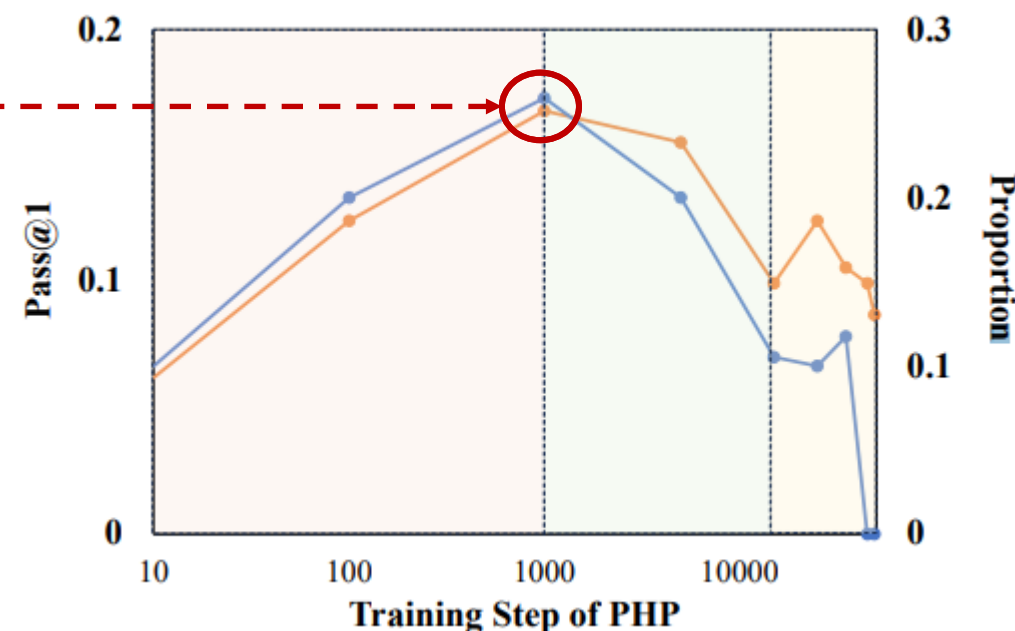
Validation Result

- **Stabilization stage**
 - The LLMs mainly work on the new languages, with the number of language transferring neurons associated with the new language continues to decrease.



- Introduction
- The Evolution Process of Multilingual Systems
- **Achieving Optimal Multilingual Performance**
- Conclusions

- The performance peak occurs during the **translation** stage in PHP and C#.
- This suggests that, for PHP and C#, **leveraging the Python system is more effective than building their own systems.**



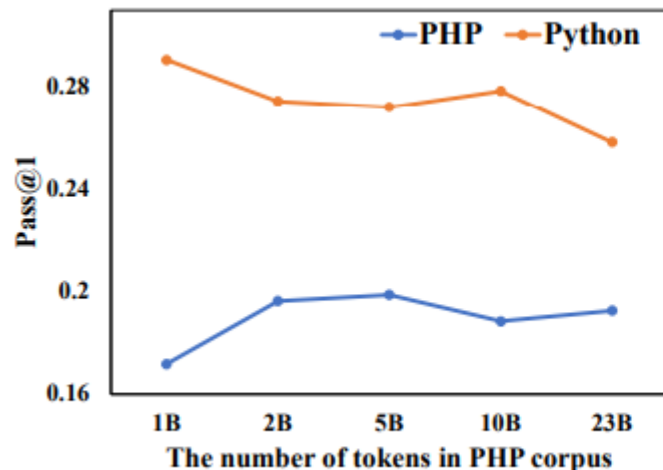
How to reach the optimal performance for different languages?

We attempt to **adjust the distribution of pre-training data** in different languages to enable different languages to achieve optimal states

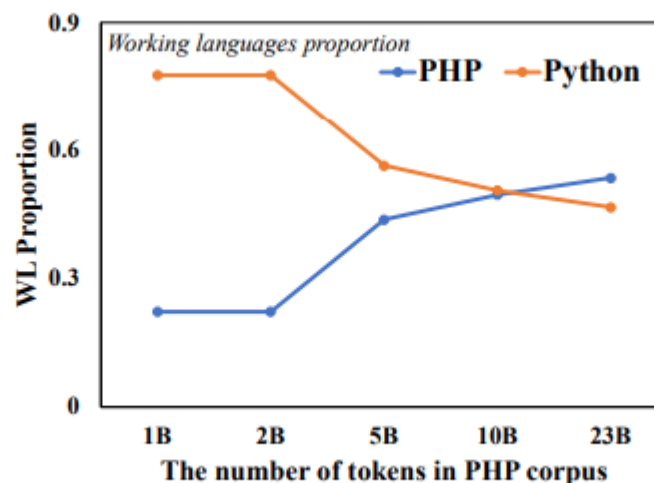
The impact of pre-training data distribution

- We pre-train the Python monolingual LLM with a mixed corpus of Python and PHP (with different number of tokens)

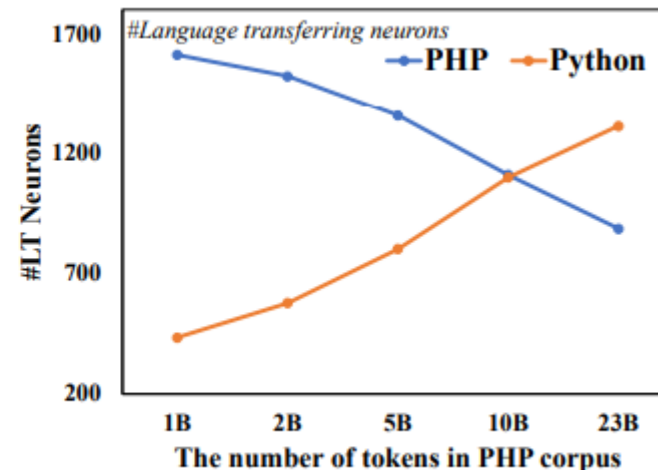
X-axis: the number of tokens of PHP corpus



HumanEval performance



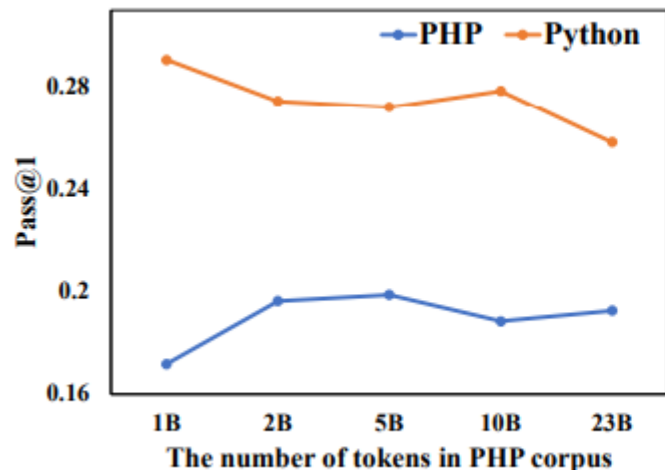
Working languages



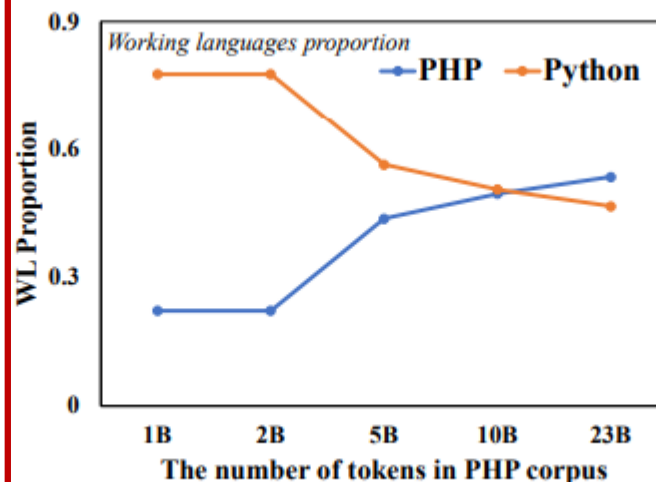
Language transferring neurons

The impact of pre-training data distribution

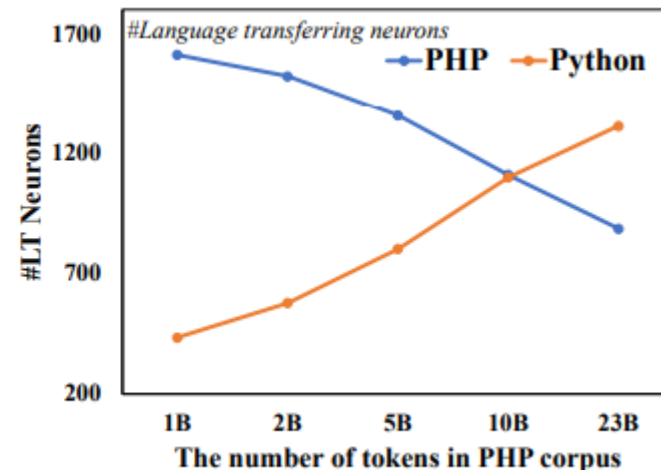
- The pre-training data distribution across different languages determines the internal state of the LLM at the Stabilization Stage.



HumanEval performance



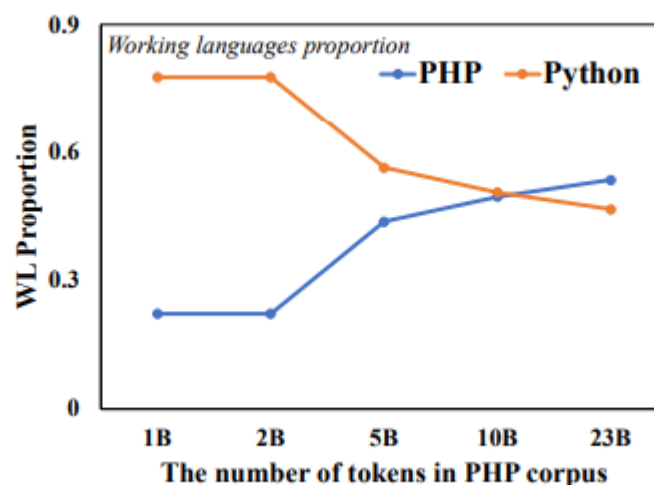
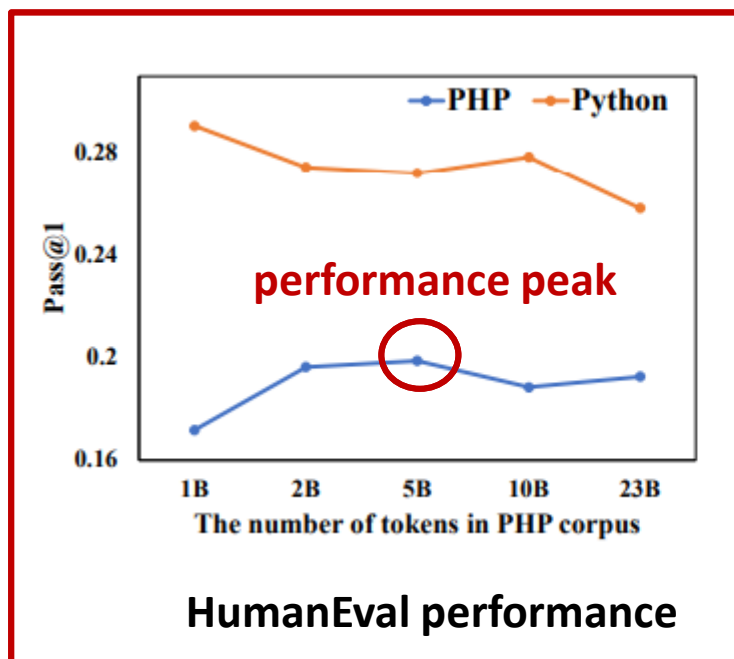
Working languages



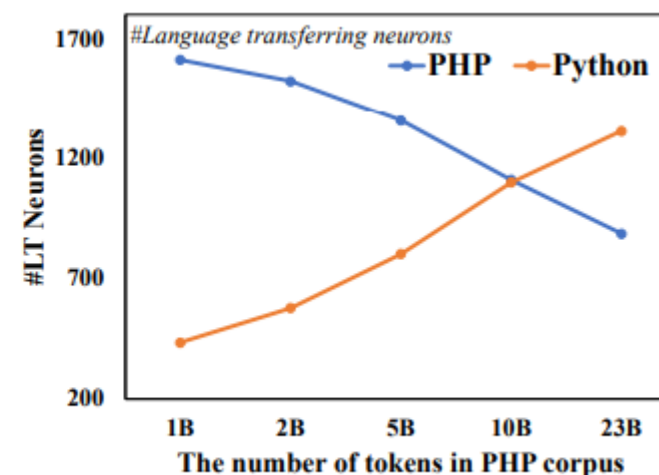
Language transferring neurons

The impact of pre-training data distribution

- Achieving the optimal pre-training data distribution across different languages is crucial for effective cross-lingual transfer.



Working languages



Language transferring neurons

Method to construct per-training corpus

- **Target**
 - determine the relationship between **overall performance** and the **pre-training data distribution** across different languages
- We use the **internal states of LLMs** as mediators since they are related to both of them.
 - the relationship between internal states and overall performance
 - the relationship between internal states and the pre-training data distribution

Method to construct per-training corpus

- The relationship between internal states and overall performance
 - We use **training loss** as a metric to measure system proportions

$$\mathcal{P}(\ell) \approx \frac{\ell - \beta}{\alpha - \beta}$$

$\mathcal{P}(\ell)$: the estimated proportion of the Python system

ℓ : training loss

α : the initial loss of LLM

β : the final loss of LLM

Method to construct per-training corpus

- The relationship between internal states and the pre-training data distribution
 - We use **the proportion of a language tokens** in the corpus as the final system proportion

$$\bar{\mathcal{P}}(\eta_i) \approx \frac{\eta_i}{\sum_j \eta_j}$$


$\bar{\mathcal{P}}(\eta_i)$: the estimated proportion of the system of language i.
 η_i : the number of tokens of language i

Method to construct per-training corpus


- **Estimate the corpus size of target language for optimal performance**

- Continual pre-training Python monolingual LLM using corpus of target language and evaluate the saved checkpoints and find the highest score.

- Find the training loss of the checkpoint with highest score and estimating the proportion of the Python system using it.


$$\mathcal{P}(\ell) \approx \frac{\ell - \beta}{\alpha - \beta}$$

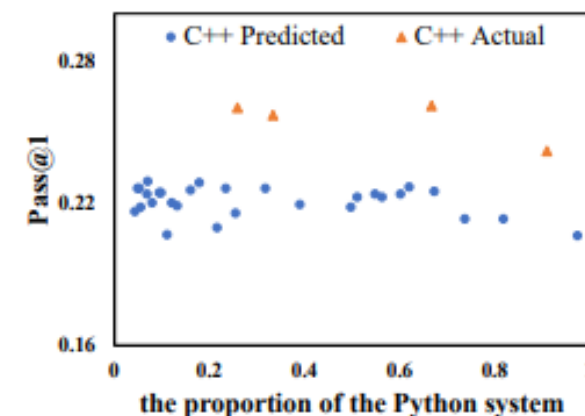
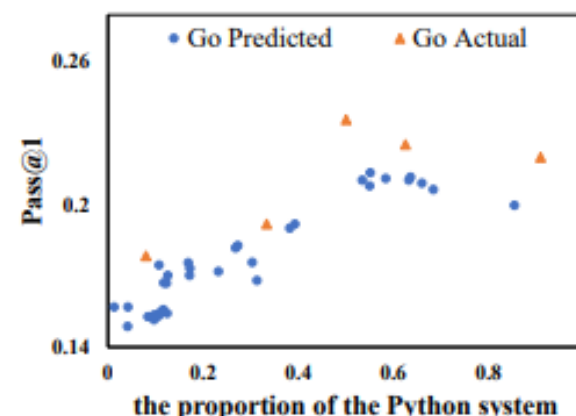
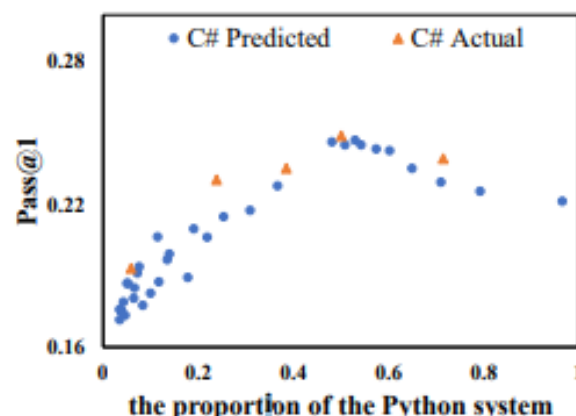
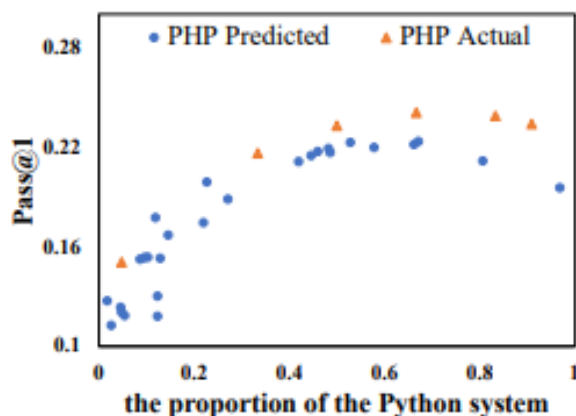
- Estimating the number of tokens in the target language corpus using the estimated proportion of the Python system


$$\bar{\mathcal{P}}(\eta_i) \approx \frac{\eta_i}{\sum_j \eta_j}$$

Experiment: validate the method

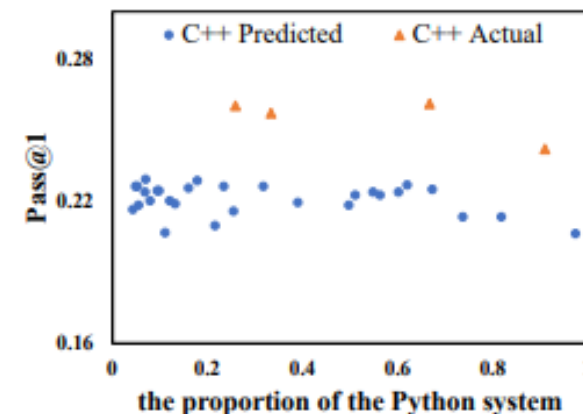
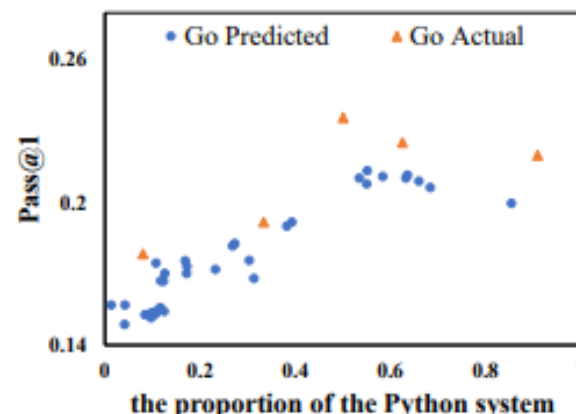
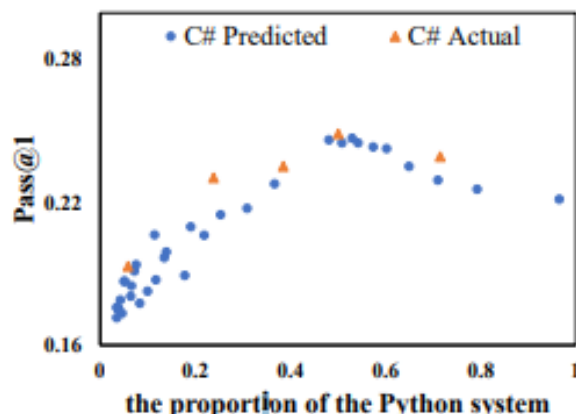
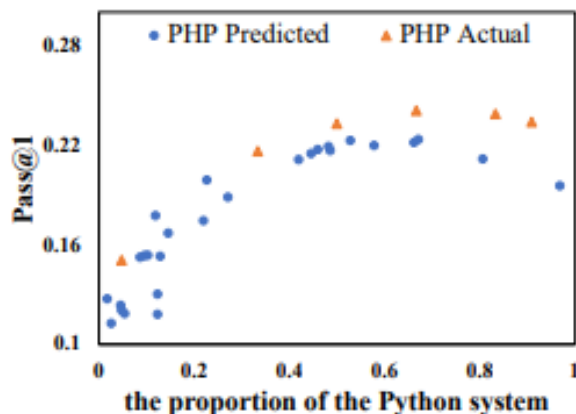
We plot the relationship between **performance** and **the proportion of Python system**

- **Blue dot**: Python system is estimated using **training loss**. (predicted result)
- **Orange dot**: Python system is estimated using **tokens in corpus**. (actual result)



Experiment: validate the method

- Despite potential numerical discrepancies, **the trend between the predicted and actual results remains consistent**, emphasizing the effectiveness of our method.



Experiment: Pre-training from scratch

- We pre-train multilingual LLMs with 1.3 billion and 6.7 billion parameters **from scratch using a mixture of corpora** from five different languages.
- Based on the predicted result, we randomly down-sampled the token quantity for PHP and C# to 5 billion and 10 billion tokens, respectively

Experiment: Pre-training from scratch

	HumanEval						MBXP					
	Python	PHP	C#	Go	C++	AVE	Python	PHP	C#	Go	C++	AVE
1B+ Models												
StarCoder-1B (Li et al., 2023b)	15.17	13.04	15.19	8.44	14.63	13.29	29.80	18.56	23.29	58.93	26.18	31.35
StarCoder-3B (Li et al., 2023b)	21.46	21.12	16.46	14.29	24.39	19.54	39.80	30.13	39.27	74.78	35.91	43.98
DeepseekCoder-1.3B (Guo et al., 2024)	29.88	23.60	27.22	20.13	31.10	26.39	52.40	45.41	48.40	78.35	55.11	55.93
Original-1.3B (ours)	28.66	<u>24.22</u>	23.42	21.43	<u>27.44</u>	25.03	41.20	33.41	<u>43.38</u>	79.24	45.39	49.81
Optimized-1.3B (ours)	32.93	26.08	29.75	16.88	<u>27.44</u>	26.61	<u>45.40</u>	46.29	40.64	77.90	<u>49.63</u>	<u>51.97</u>
6B+ Models												
StarCoder-7B (Li et al., 2023b)	28.37	24.22	27.85	17.53	25.61	24.72	46.20	42.36	45.66	71.88	43.64	49.95
StarCoder2-7B (Lozhkov et al., 2024)	35.40	32.30	39.24	22.08	34.76	32.76	53.13	55.02	54.79	82.37	55.61	60.18
CodeLlama-7B (Rozière et al., 2024)	33.50	24.22	34.81	20.13	27.44	28.02	49.80	43.89	48.40	72.32	48.63	52.61
DeepseekCoder-6.7B (Guo et al., 2024)	46.95	37.89	47.47	31.82	44.51	41.73	65.80	66.16	66.67	89.78	67.58	71.20
Original-6.7B (ours)	<u>46.95</u>	<u>38.51</u>	39.24	26.62	40.24	38.31	59.00	57.86	61.19	<u>83.04</u>	60.60	64.34
Optimized-6.7B (ours)	48.17	41.61	<u>44.94</u>	<u>31.17</u>	<u>43.90</u>	41.96	<u>63.60</u>	<u>59.83</u>	<u>61.42</u>	<u>81.03</u>	<u>64.84</u>	<u>66.14</u>

- Compared to the original pre-training corpus, the corpus optimized with our method exhibits a significant average performance improvement.

Experiment: Pre-training from scratch

	HumanEval						MBXP					
	Python	PHP	C#	Go	C++	AVE	Python	PHP	C#	Go	C++	AVE
1B+ Models												
StarCoder-1B (Li et al., 2023b)	15.17	13.04	15.19	8.44	14.63	13.29	29.80	18.56	23.29	58.93	26.18	31.35
StarCoder-3B (Li et al., 2023b)	21.46	21.12	16.46	14.29	24.39	19.54	39.80	30.13	39.27	74.78	35.91	43.98
DeepseekCoder-1.3B (Guo et al., 2024)	29.88	23.60	27.22	20.13	31.10	26.39	52.40	45.41	48.40	78.35	55.11	55.93
Original-1.3B (ours)	28.66	24.22	23.42	21.43	27.44	25.03	41.20	33.41	43.38	79.24	45.39	49.81
Optimized-1.3B (ours)	32.93	26.08	29.75	16.88	27.44	26.61	45.40	46.29	40.64	77.90	49.63	51.97
6B+ Models												
StarCoder-7B (Li et al., 2023b)	28.37	24.22	27.85	17.53	25.61	24.72	46.20	42.36	45.66	71.88	43.64	49.95
StarCoder2-7B (Lozhkov et al., 2024)	35.40	32.30	39.24	22.08	34.76	32.76	53.13	55.02	54.79	82.37	55.61	60.18
CodeLlama-7B (Rozière et al., 2024)	33.50	24.22	34.81	20.13	27.44	28.02	49.80	43.89	48.40	72.32	48.63	52.61
DeepseekCoder-6.7B (Guo et al., 2024)	46.95	37.89	47.47	31.82	44.51	41.73	65.80	66.16	66.67	89.78	67.58	71.20
Original-6.7B (ours)	46.95	38.51	39.24	26.62	40.24	38.31	59.00	57.86	61.19	83.04	60.60	64.34
Optimized-6.7B (ours)	48.17	41.61	44.94	31.17	43.90	41.96	63.60	59.83	61.42	81.03	64.84	66.14

- Our method remains effective even when applied to an LLM with 6.7 billion parameters, indicating the universality of our approach.

- Introduction
- The Evolution Process of Multilingual Systems
- Achieving Optimal Multilingual Performance
- **Conclusions**

- We propose the Babel Tower Hypothesis to explain the evolution process of multilingual LLM during pre-training: during the learning process of large language models (LLMs), multiple languages initially share a single knowledge system dominated by a primary language and then gradually shift to developing multiple language-specific knowledge systems
- We validate the Babel Tower Hypothesis in code LLMs by employing two methods to probe and analyze the internal states of LLMs during pre-training.
- We design an effective method to construct a pre-training corpus with optimal distribution across different languages, providing a guide for the pre-training process.

Thanks!