A Study on the Impact of Common Code Structures on CodeParrot's Autocompletion Performance

Razvan-Mihai Popescu
R.Popescu-3@student.tudelft.nl

Supervisors: Maliheh Izadi, Jonathan Katzy
Professor: Arie van Deursen

01 Introduction

Large language models (LLMs), particularly those based on deep learning techniques, such as transformer models [1], have shown significant promise in improving the accuracy and effectiveness of code completion.

However, most models are only trained in **one language** or **multiple very common languages** at the same time. Thus, many of the plugins being developed, such as GitHub Copilot [2], will function best with popular languages such as Python or Java, but may struggle to operate on low-resource languages such as Julia, Go, or Kotlin.

Furthermore, in the context of code completion tasks, the significance of **common code structures (CCS)** on the performance of LLMs remains an understudied aspect. Their presence might significantly impact the attention mechanism and the performance of code completion models.

This study aims to answer whether ***common code structures relate to the depth of the first correct completion (DoFCC) of the CodeParrot language model [3].***

02 Methodology

The CCS can be categorized into distinct groups, including control structures, functions, classes, objects, exceptions, data structures, and input/output (I/O) structures. To understand their implications, we divide our analysis into two parts: ***the tuned lens investigation*** and ***the attention investigation.***

We are using a multi-language test bench comprising languages such as Python, Java, C++, Kotlin, Julia, and Go. Out of these, Kotlin is the only language CodeParrot was not trained on. Each dataset was extracted from **The Stack dataset** and contains 512 source code files.

For the tuned lens investigation, we start by creating collections of the most CCS for each language. CodeParrot is then used to perform token completion, while the **tuned-lens method** [4] is applied to provide insights into the model's internal representations. **The DoFCC** is used as the evaluation metric, indicating the layer where the intermediate result aligns with the expected token. Lastly, various statistical techniques are utilized to quantify intermediate model results.

In the attention investigation, we collect attention heads for both common and uncommon code structures (UCS). Afterward, diverse statistical methods are employed to examine the aspect of **null attention** [5], which refers to attention heads where the first token receives at least half of the attention score.

Finally, we analyze the **correlation** between the results of the two investigations, aiming to comprehend the overall impact and significance of these CCS and to understand the decision-making process of the model.

03 Results

Our findings show that the influence of the CCS on completion performance is apparent across all six languages, particularly in **high-resource languages** such as Java, Python, and C++, as can be seen in *Figure 1.*

In low-resource languages, Go demonstrates remarkable performance comparable to Java, compensating for the scarcity of training data due to its minimalist syntax, as observed in *Figure 2.* Conversely, the influence of typical structures is reduced in the context of Julia, as illustrated in *Figure 3.* Despite the lack of training data, Kotlin achieves similar results to Julia, due to its syntactic similarity to Java.

Additionally, **maintaining proper indentation** and **adhering to whitespace conventions** leads to a higher completion accuracy. This is depicted in *Figure 4*, where common tokens such as *"_return" or "_if"* obtain a DoFCC of just one in Julia.

Moreover, *Figure 5* showcases the consistent completion performance observed for **print statements** in Python. The **pattern of consecutive CCS** was observed across print statements in all languages. It emerges when clusters of CCS appear successively, leading to better predictions. This results in fluctuations in the DoFCC values, which can be seen at the beginning of print statements in C++, as depicted in *Figure 6.*

Both CCS and UCS achieved **consistent and similar attention results** for all languages. An important pattern can be seen in *Figure 7*, where the majority of null attention heads (NAHs) are found in the upper layers rather than the lower layers. This correlates with the average DoFCC for CCS and suggests that the NAHs in the final layers do not provide any relevant contextual information for the predictions.

04 Conclusion

In light of these findings, CCS demonstrate a considerable impact on CodeParrot's completion performance for both high- and low-resource languages due to their **frequent occurrence, consistent syntax, clear semantics, and contextual clues.**

Further, both CCS and UCS illustrate **similar attention results across all six languages. A robust correlation** was identified between the DoFCC of CCS and the limited occurrence of NAHs in the network's initial layers. Our observations indicated that most NAHs originating from the upper layers lack meaningful contextual information that contributes to the predictions.

At the same time, this presents a **promising avenue for future investigation.** By studying these CCS in controlled environments, we can reinforce our observations and enhance the significance of our findings.

Related literature

[1] Vaswani *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017

[2] Mark Chen *et al.* Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021

[3] Xu *et al.* A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 2022

[4] Belrose *et al.* Eliciting latent predictions from transformers with the tuned lens. *ArXiv*, abs/2303.08112, 2023

[5] Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 158–165, 2020

Figure 1

The mean and standard deviation of the DoFCC for CCS and UCS in Java

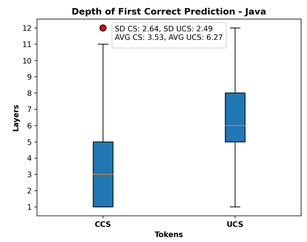


Figure 2

The mean and standard deviation of the DoFCC for CCS and UCS in Go

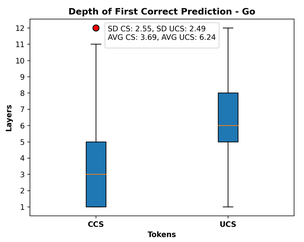


Figure 3

The mean and standard deviation of the DoFCC for CCS in Julia

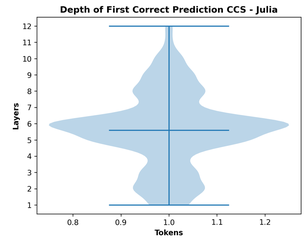


Figure 4

Low outliers of CCS in Julia

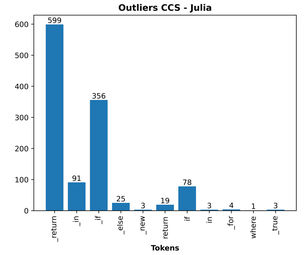
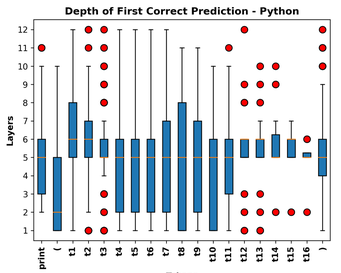


Figure 5

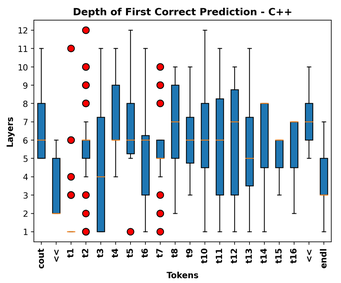
The mean and standard deviation of the DoFCC for print statements in Python



Standard Deviation	Frequency
'print': 2.12	'print': 325
'(': 2.27	'(': 325
't1': 2.57	't1': 304
't2': 2.34	't2': 263
't3': 2.26	't3': 227
't4': 2.68	't4': 207
't5': 2.7	't5': 187
't6': 2.64	't6': 157
't7': 2.89	't7': 124
't8': 3.05	't8': 111
't9': 2.81	't9': 95
't10': 2.66	't10': 76
't11': 2.53	't11': 67
't12': 2.13	't12': 45
't13': 2.29	't13': 32
't14': 1.95	't14': 20
't15': 1.59	't15': 11
't16': 1.17	't16': 8
')': 2.03	')': 325

Figure 6

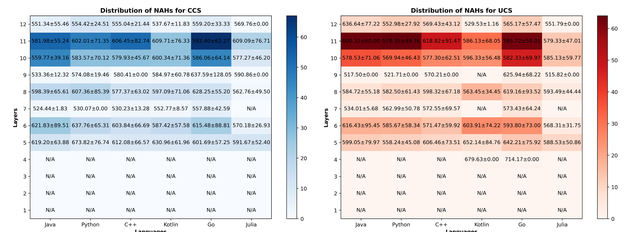
The mean and standard deviation of the DoFCC for print statements in C++



Standard Deviation	Frequency
'cout': 1.51	'cout': 131
'<<': 1.27	'<<': 131
't1': 1.41	't1': 131
't2': 2.21	't2': 126
't3': 3.25	't3': 104
't4': 1.77	't4': 88
't5': 2.06	't5': 46
't6': 2.49	't6': 40
't7': 2.21	't7': 33
't8': 2.57	't8': 27
't9': 2.09	't9': 20
't10': 2.86	't10': 15
't11': 3.09	't11': 12
't12': 3.42	't12': 6
't13': 3.25	't13': 6
't14': 3.3	't14': 3
't15': 1.41	't15': 3
't16': 2.36	't16': 3
'<<': 1.08	'<<': 131
'endl': 1.08	'endl': 131

Figure 7

The frequency, mean, and standard deviation of null attention heads per layer and within each language, for both CCS and UCS



TUDelft