LR 파싱을 이용한 LLM 기반 코드 완성에서 ChatGPT 3.5 와 Llama 3 의 비교 분석

Md Monir Ahammod Bin Atique[1], Kwanghoon Choi[2]
1 전남대학교 인공지능융합학 석사과정, 2 전남대학교 인공지능융합학과 교수
monir024@jnu.ac.kr, kwanghoon.choi@jnu.ac.kr

A Comparative Analysis of ChatGPT 3.5 and Llama 3 in LLM-Based Code Completion Using LR Parsing

Md Monir Ahammod Bin Atique, Kwanghoon Choi
Dept. of Artificial Intelligence Convergence, Chonnam National University, South Korea

Abstract

Modern integrated development environments (IDEs) rely on code completion as a key feature to enhance coding efficiency and streamline the developer workflow. Traditional approaches to code completion have often relied on rule-based techniques, static ranking, and prefix-based filtering, which pose challenges in terms of usability and efficiency. Recent research has introduced LR-parsing-based approaches that generate structural candidate suggestions by leveraging language syntax and open-source programs, but they often require manual refinement. Meanwhile, recent advancements in large language models (LLMs) have significantly amplified predictive performance in code completion tasks. However, despite these progress, the impact of LLM selection on LR-parsing based syntax-aware code generation remains underexplored. In this study, we conduct a comparative analysis of the performance and impact of two prominent LLMs, ChatGPT 3.5 and Llama 3, within an LR parsing-based code completion framework. Our experiments, evaluated using SacreBLEU and SequenceMatcher accuracy metrics, reveal that ChatGPT 3.5 achieves higher accuracy than Llama 3, underscoring the importance of selecting an appropriate LLM for enhanced code completion. These findings highlight the role of model selection in LLM-based code completion using LR parsing. Future research could extend this comparative analysis to a broader range of LLMs.

1. Introduction

Code completion is a fundamental feature in modern IDEs, significantly promoting developer productivity by reducing keystrokes, minimizing syntax errors, and improving efficiency in writing syntactically correct code. Traditional code completion methods primarily rely on prefix-based filtering and static ranking, often producing extensive suggestion lists with limited contextual awareness and without effective ranking. To address these limitations, several approaches have been taken by researchers. One novel approach, based on syntax analysis and LR parsing theory [1], has been proposed. On the other hand, recent growth in LLMs have transformed code completion strategies by leveraging vast training data to generate sophisticated, accurate, context-aware, and probabilistic predictions. Notably, models such as OpenAI's ChatGPT (3.5 Turbo) and Meta AI's Llama 3 have recently demonstrated substantial capabilities in generating syntactically correct and semantically meaningful code suggestions. However, despite these advancements, ensuring broader structural correctness, including adherence to language constraints and maintaining logical consistency, remains a challenge in LLM based code completion.

LR parsing, a well-established technique in compiler theory, has been effectively employed in code completion to generate structured completion candidates (structural candidate). Sasano and Choi [2,3] formally defined structural code completion candidates (γ) for a prefix ($\alpha\beta$) in a sentential form. If an LR grammar contains a production $A \rightarrow \beta\gamma$ allowing $\beta\gamma$ to reduce to nonterminal A, then γ qualifies as structural candidates. The concept of structural candidates for code completion within the framework of LR parsing is explored in detail by Sasano and Choi in [2]. In a subsequent study, Choi et al. [4] introduced a ranking mechanism for these structural candidates based on pre-analyzed frequencies of their occurrences in open-source projects, but their study left usability concerns unresolved. More recently, in our previous study [5], we proposed LLMs, such as ChatGPT, with LR parsing-based structural candidates. This approach presents a promising solution for syntax-aware code completion, enhancing both productivity and usability. However, the comparative effectiveness of different LLMs in LR parsing-based code completion remains unexplored, leading to a key research question: Which model, ChatGPT or Llama, demonstrates better performance in LR parsing-based code generation?

To address the research question, this paper presents a comparative experimental study analyzing the performance of ChatGPT 3.5 and Llama 3 in LLM-based code completion using LR parsing. We evaluated their performance using SacreBLEU and SequenceMatcher accuracy metrics to determine the impact of model selection on syntax-aware code generation with LR parsing. Our key findings indicate that ChatGPT 3.5 outperforms Llama 3, achieving higher accuracy in syntactically correct code completions in experiments using Microsoft Small Basic and C languages. This reinforces the importance of selecting an appropriate LLM to optimize code completion within LR-structured candidates. This study makes the following key contribution:

- We assess the performance of different LLMs, such as ChatGPT 3.5 and Llama 3, in LR parsing-based code completion, highlighting the impact of model selection on accuracy.
- Our findings indicate that choosing the ChatGPT model over Llama provides advantage in achieving higher accuracy for LLM-based code completion using LR parsing.

The significance of our study lies in its contribution to understanding the role of LLM selection in syntax-aware code completion and the interplay between LR structural candidates and probabilistic language models. Our findings offer valuable insights into the application of LLMs in code completion, benefiting IDE developers and researchers aiming to optimize code completion strategies. Given these insights, future research could expand upon this study by investigating alternative methods for integrating structured parsing techniques. This paper is organized as follows: Section 2 covers related work, Section 3 presents our overall system, Section 4 discusses the results and analysis, and Section 5 provides the conclusion along with future directions.

2. Related Work

The integration of LLMs into code completion tools has garnered significant attention in recent research. OpenAI's ChatGPT has been extensively utilized for code generation and completion tasks. Studies have demonstrated that prompt engineering can substantially enhance ChatGPT's code generation performance, highlighting the model's adaptability to various coding scenarios [6]. Additionally, in their study, empirical evaluations have assessed ChatGPT's effectiveness in code generation, program repair, and code summarization, providing insights into its practical applications and limitations in software engineering.

In parallel, Meta AI introduced Code Llama, a code-specialized version of the Llama 3 model, designed to generate and discuss code. Trained on a diverse dataset, it supports multiple programming languages, including Python, C++, Java, and more, aiming to enhance developer workflows by providing efficient code generation and completion capabilities [7]. Comparative analyses have shown that it outperforms other models in specific tasks, such as OpenAPI code completion, indicating its potential superiority in certain coding applications [8].

Despite these advancements, direct comparative studies between ChatGPT and Llama-based models in the context of LR parsing-based code completion remain limited. This study aims to address this gap by empirically evaluating the performance of ChatGPT 3.5 and Llama 3 within this specific LR-parsed framework, providing insights into their relative effectiveness in generating accurate and syntactically correct code completions.

3. Methodology

3.1 Experimental Setup

To evaluate the performance of ChatGPT 3.5 and Llama 3 in LR parsing-based code completion, we conducted experiments on two programming languages: Microsoft Small Basic (SB) and C11. These languages were chosen for their distinct syntax structures and their relevance in introductory programming and system-level programming, respectively. Our experimental workflow consists of two main phases: the collecting and ranking phase (offline) and the query phase (online) which is depicted in Figure 1. The offline phase serves as the foundation of this research and has been detailed in previous work [4], while the online phase is the primary focus of this study. The online phase provides an efficient code completion system by leveraging LR parsing and LLM-based candidate code generation.

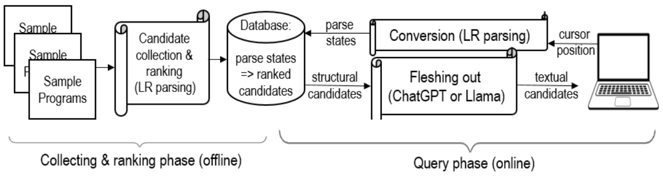

Figure 1: Overview of our system workflow

The experimental setup involved the following steps:

1. Candidate collection & ranking by LR parsing (offline): The training set, collection of samples undergoes LR parsing to generate structural candidates and then ranked in this phase. The training set consists of 3,701 Small Basic programs collected from its community and 412 C11 programs sourced from open-source software repositories.

2. Database storage: The parsed structural candidates are stored in a database, where they are ranked based on their occurrence frequency in the training set. The database [4] maintains a mapping between parse states and their ranked candidates, enabling efficient retrieval. For instance, in the database for State 0, several structural candidates are stored along with their occurrence frequencies. Below, we present three ranked candidates for State 0 as an example:

   [ID=Expr] : 422

   [ID.ID= Expr] : 399

   [ID.ID(Exprs)] : 246

Among these, one structural candidate is the correct choice for parse state 0, referred to as the ideal structural candidate (e.g., ID.ID(Exprs) in this example) for a specific cursor position in a test program. In this case, ID represents a terminal identifier, parentheses () are also terminal symbol, and Exprs is a non-terminal expression. Terminal symbols, also known as tokens, serve as the fundamental building blocks of a language, while nonterminal symbols, or syntactic variables, represent sets of strings composed of terminal symbols.

3. Parsing & candidate retrieval: Upon receiving a user query, the system converts the cursor position into a parse state using the LR parsing technique and retrieves the corresponding ranked structural candidates from the database. To evaluate the system, we used a testing set comprising 27 Small Basic programs from its tutorial materials and 106 C11 programs from the exercises in The C Programming Language by Kernighan and Ritchie.

4. Model invocation: The selected ideal structural candidates are used to construct completion prompts, which are first processed by ChatGPT (gpt-3.5-turbo-0125) and subsequently by Llama 3 (llama-3.1-8b-instant) model. Since the correct (ideal) LR structural candidates are always provided to the LLM in this study, this represents an optimal approach, ensuring the highest possible accuracy in code completion.

5. Final code suggestions: Our system automatically constructs prompts with structured candidates, enabling LLMs to generate relevant textual suggestions. These generated suggestions are then finally presented to the user at the specific cursor position.

6. Evaluation metrics: The generated textual code completions were evaluated using two primary metrics:

   SacreBLEU (%): Measures the n-gram similarity between the generated and reference code, ensuring token-level accuracy [9].

   SequenceMatcher (%): Assesses character-level alignment to determine sequence similarity, identifying the longest matching subsequences for a similarity ratio calculation [10].

3.2 Prompt Engineering for ChatGPT 3.5 and Llama 3

In the fourth step of our proposed system, prompts containing ideal structural candidates were constructed to generate completion responses for all possible cursor positions in the test set. Each prompt template consists of an incomplete code prefix, a selected ideal structural candidate for each parse state (cursor position), and an instruction to the LLM for performing code completion, as illustrated in Figure 2. This template is then fed into the ChatGPT 3.5 and Llama 3 models. Figure 3 presents example prompts for Microsoft Small Basic and C. Such prompts were crafted for all test programs during the online phase. Notably, our system remains language-agnostic by instantiating prompt templates with parameters tailored to each specific programming language. Figure 4 presents a comparison of ChatGPT 3.5 and Llama 3 responses to a prompt in Microsoft Small Basic. ChatGPT 3.5 achieves perfect scores with a SacreBLEU of 100% and SequenceMatcher similarity of 100%, exactly matching the expected output. Here, it follows the candidate structure 'ID(Expr)'. In contrast, Llama 3 scores significantly lower, with a SacreBLEU of 33.33% and a SequenceMatcher similarity of 43.90%, only slightly following the structural candidate.

Prompt Template with Ideal Structural Candidate Guidance

```
1: This is the incomplete {Name of Programming Language} code:
2: {Program Prefix}
3: {Suggested Ideal Structural Candidate}
4: Complete the {Suggested Ideal Structural Candidate} part of the code
5: in the {Name of Programming Language}.
6: Just show your answer in place of {Suggested Ideal Structural Candidate}.
```

Figure 2: Prompt engineering with ideal structural candidate

Example of Prompt with Ideal Structural Candidate Guidance in Microsoft Small Basic Language

```
1: This is the incomplete Microsoft Small Basic programming
2: language code:
3: number = 100
4: While (number > 1)
5:                    TextWindow.
6:                                                'ID(Expr)'
7: Complete the 'ID(Expr)' part of the code in the Microsoft Small Basic
8: programming language. Just show your answer in place of 'ID(Expr)'.
```

Example of Prompt with Ideal Structural Candidate Guidance in C Language

```
1: This is the incomplete C programming language code:
2: int main(void)
3: {
4:     char s[1000];
5:     int i = 0;
6:     int loop = 1;
7:                 'while (expression) scoped statement'
8: Complete the 'while (expression) scoped_statement' part of the code
9: in the C programming language. Just show your answer in place of
10: 'while (expression) scoped_statement'.
```

Figure 3: Prompt examples for Microsoft SmallBasic and C

Comparative Evaluation of the Example in Figure 3 in Microsoft Small Basic Language Using ChatGPT 3.5 and Llama 3

```
ChatGPT 3.5 Response: WriteLine(number)
Response Evaluation:
    SacreBLEU (%) score: 100
    SequenceMatcher(%) similarity precision: 100
Llama 3 Response: TextWindow.WriteLine
Response Evaluation:
    SacreBLEU (%) score: 33.333
    SequenceMatcher(%) similarity precision: 43.902
Actual Textual Answer: WriteLine(number)
```

Figure 4: Comparative Evaluation of LLM Responses in SB

4. Results and Discussion

4.1 Comparative Results Analysis

Table 1 presents the comparative results of code completion performance between ChatGPT 3.5 and Llama 3 for SB and C11, using ideal structural candidate guidance using LR parsing technique for all test programs in terms of average SacreBLEU and SequenceMatcher.

Table 1: Code completion experiment results with ideal structural candidate guidance using different LLMs.

Language	**LLM Type**	**SacreBLEU (%)**	**SequenceMatcher (%)**
SB	ChatGPT	43.856	42.618
	Llama 3	29.086	30.374
C11	ChatGPT	25.173	26.537
	Llama 3	15.290	16.913

The results indicate that ChatGPT consistently outperforms Llama 3 in code completion accuracy, emphasizing that the choice of LLM plays a crucial role in optimizing syntax-aware code generation.

4.2 Discussion

Our key observations from the experimental results are as follows:

Higher accuracy with ChatGPT: ChatGPT demonstrates significantly better performance than Llama 3, achieving an improvement of nearly 10% to 15% in both SacreBLEU and SequenceMatcher scores. Specifically, ChatGPT achieves a SacreBLEU improvement of 14.77 for Small Basic and 9.883 for C11. Similarly, the SequenceMatcher scores show an increase of 12.244 for Small Basic and 9.624 for C11. These results indicate that ChatGPT generates more precise and structurally aligned code completions compared to Llama 3 across different programming languages.

Significance of model selection: These results emphasize the importance of selecting the right LLM for syntax-aware code completion. While both models benefit from structured candidate guidance, ChatGPT's architecture appears better suited to incorporating explicit structural candidates into its predictions.

5. Conclusion and Future Work

This study presents a comparative experimental report evaluating the performance of ChatGPT 3.5 and Llama 3 in LR parsing-based code completion. Our findings demonstrate that selecting ChatGPT offers a clear advantage in improving accuracy, as measured by SacreBLEU and SequenceMatcher scores. This highlights the importance of choosing the right LLM for optimizing syntax-aware code completion.

Future research should focus on expanding the comparative analysis to include a wider variety of LLMs to assess their effectiveness in different programming environments. Exploring the integration of additional fine-tuned LLMs could further refine accuracy. Additionally, assessing real-world developer usability and efficiency metrics will provide deeper insights into practical adoption. This research lays the groundwork for further advancements in selecting and refining LLMs for high-accuracy code completion.

Acknowledgments

This work was supported by Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP-2023-RS-2023-00256629). This work was also partially supported by the Korea Internet & Security Agency (KISA) - Information Security College Support Project.

References

[1] Aho, Alfred, Monica Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers—Principles, Techniques, and Tools.* Addison Wesley, 2006.

[2] Sasano, Isao, and Kwanghoon Choi. "A text-based syntax completion method using lr parsing." In *Proceedings of the 2021 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, New York, NY, USA, 2021, pp. 32-43.

[3] Sasano, Isao, and Kwanghoon Choi. "A text-based syntax completion method using LR parsing and its evaluation." *Science of Computer Programming* 228 (2023): 102957.

[4] Choi, Kwanghoon, Sooyeon Hwang, Hyeonah Moon, and Isao Sasano. "Ranked Syntax Completion With LR Parsing." In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, New York, USA 2024, pp. 1242-1251.

[5] Atique, Md Monir Ahammod Bin, Kwanghoon Choi, Isao Sasano, and Hyeon-Ah Moon. "Improving LLM-based Code Completion Using LR Parsing-Based Candidates." In *CEUR Workshop Proceedings*, 2024, vol. 3754, pp. 1-6. CEUR-WS.

[6] Liu, Jiawei, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation." *Advances in Neural Information Processing Systems* 36: 21558-21572, 2023.

[7] Meta AI. "Introducing Code Llama, a State-of-the-Art Large Language Model for Coding." Meta AI Blog, August 24, 2023. https://ai.facebook.com/blog/code-llama-large-language-model-coding/.

[8] Caumartin, Genevieve, Qiaolin Qin, Sharon Chatragadda, Janmitsinh Panjrolia, Heng Li, and Diego Elias Costa. "Exploring the Potential of Llama Models in Automated Code Refinement: A Replication Study." *arXiv preprint arXiv:2412.02789*, 2024.

[9] Post, Matt. "A call for clarity in reporting BLEU scores." arXiv preprint arXiv:1804.08771 (2018).

[10] Foundation, P.S., 2023. Difflib — Helpers for computing deltas. Available at https://docs.python.org/3/library/difflib.html.