Jetbrains AI Code Completion Internship

Topic: Evaluation of LLM predictions on FIM datasets using popular metrics

Author: Krzysztof Nowak

Abstract

This paper investigates the effectiveness of code completion using the Fill-in-the-Middle (FIM) method by implementing and evaluating a code completion system on personal project repositories. The research employs a systematic approach to generate a dataset of 20-50 code completion examples by splitting code segments into three parts: prefix, middle (target completion), and suffix, simulating realistic cursor positions during coding, using PSM and SPM methods.
The study utilizes the StarCoder model, an open-source code completion system, to generate predictions for the missing middle segments. I implement a comprehensive evaluation framework that combines both manual assessment and automated metrics. The automated evaluation includes exact match scoring, chrF (character n-gram F-score), and additional custom metrics to capture various aspects of completion quality.

1. Introduction

Code completion has become an integral part of modern software development, significantly enhancing programmer productivity and reducing coding errors. The Fill-in-the-Middle (FiM) method represents a novel approach to this challenge, offering a more context-aware solution compared to traditional prefix-only completion methods.

Fill-in-the-Middle (FiM) differs from traditional approaches by allowing bidirectional context utilization during text generation. Unlike left-to-right autoregressive models, which can only use previous context, FiM can leverage both prefix and suffix information to generate more contextually appropriate completions. Compared to Masked Language Modeling (MLM) used in BERT, FiM is better suited for generative tasks as it can handle variable-length infills while MLM typically predicts single tokens (at a time). While insertion-based methods also work with gaps, FiM's direct approach to middle-filling is more efficient and produces more coherent results. FiM also shows advantages over Prefix-to-Prefix (P2P) methods in scenarios requiring precise contextual alignment between the beginning and end of text. However, FiM may require more computational resources than simpler left-to-right models and might not be necessary for all applications where sequential generation is sufficient.

2. Dataset Creation

The dataset creation process involves segmenting code snippets into three distinct components based on cursor position:

- PREFIX (the code segment preceding the cursor position),
- MIDDLE (the ground truth completion),
- SUFFIX (the remaining code following the completion).

For example, given the code "my_dog = Dog(name='Alex', age = 15)" with cursor position at "my_dog = D", the segmentation yields PREFIX: "my_dog = D", MIDDLE: "og", and SUFFIX: "(name='Max', age = 5)". The results are stored in JSONL format following the pattern **<fim_prefix>{prefix}<fim_suffix>{suffix}<fim_middle>{middle}** which is a **PSM** format. I will also create datasets in **SPM** format (**<fim_suffix>{suffix}<fim_prefix>{prefix}<fim_middle>{middle}**) to compare those two techniques. The segmentation process randomly selects prefix, suffix, and middle components, with each part corresponding to approximately ⅓ of the document. Additional attributes are included in each record: original text for quick comparison, path_file for source identification, and split_index indicating the iteration number over the same data, enabling generation of multiple examples from each file. The implementation focuses on Python and text files (**Hamlet - William Shakespeare**) derived from two repositories to ensure diverse examples, with text files serving as an additional method to evaluate model robustness.

3. **Code completion Model**

TinyStarCoder model was chosen as a middle-ground between run-time and efficiency. Model was trained on Python data from StarCoderData for ~6 epochs which amounts to 100B tokens. It's overall a **164M** parameter model.

4. **Metrics used to evaluate the model**

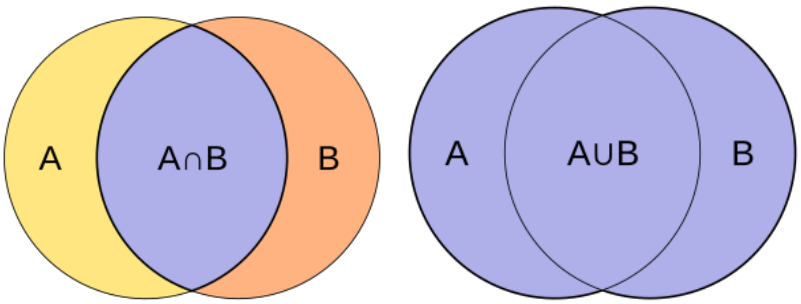
1) CHARF (Character F-score)
CHARF measures the character-level similarity between predicted and ground truth sequences. We can derive a formula to calculate CHARF metric:

*CHARF = 2 * (precision * recall) / (precision + recall)*

Key characteristics of CHARF include:

- Character-level granularity
- Sensitive to exact character matches
- Range: [0,1], where 1 indicates perfect match

2) Jaccard Similarity
Measures the overlap between predicted and ground truth token sets.



Intersection and union of two sets A and B
Source: https://en.wikipedia.org/wiki/Jaccard_index

Jaccard = |A ∩ B| / |A ∪ B|

where A and B are the sets of tokens in predicted and ground truth sequences

Key characteristics:

- Token-level evaluation
- Order-independent
- Robust to slight variations
- Range: [0,1], where 1 indicates identical sets

3. ROUGE Score (Recall-Oriented Understudy for Gisting Evaluation)
ROUGE is a set of metrics for evaluating automatic summarization and machine translation.
Variants:

- ROUGE-N: N-gram overlap
- ROUGE-L: Longest Common Subsequence

Key characteristics:

- Multiple granularity levels
- Considers sequence order
- Range: [0,1], higher values indicate better matching

4. BLEU (Bilingual Evaluation Understudy)
Originally designed for machine translation, adapted for code completion evaluation.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where:

- BP: Brevity penalty
- wn: n-gram weights
- pn: n-gram precision

Key characteristics:

- N-gram based evaluation
- Considers precision at multiple levels
- Penalizes both too-short and too-long predictions
- Range: [0,1], where 1 indicates perfect match

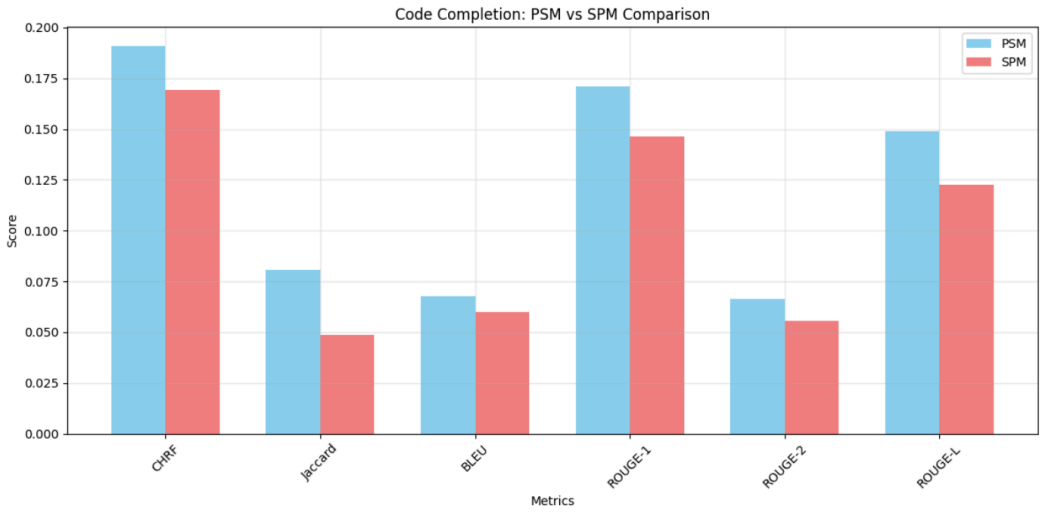
5. Exact Match
Exact Match (EM) is the strictest evaluation metric that checks if the predicted text matches the reference text perfectly - character by character, word by word.

5. Predicted examples evaluation and results

Analysis of the evaluation metrics indicates that the current segmentation approach, which processes entire files, results in relatively low average performance scores. This can be attributed to the inherent complexity of predicting substantial code segments with high accuracy. The current implementation maintains an equal distribution **ratio of ⅓** for each component (PREFIX, MIDDLE, SUFFIX). However, the model's predictive performance could potentially be enhanced through strategic adjustment of these segment proportions.
By reducing the MIDDLE segment size while proportionally increasing the PREFIX and SUFFIX components, we could provide the model with expanded contextual information. This modification to the segmentation ratio would likely yield improved prediction accuracy, as the model would benefit from:

- Enhanced contextual understanding through larger PREFIX and SUFFIX segments
- Reduced complexity of the prediction task due to shorter MIDDLE segments
- More focused prediction scope with increased surrounding context

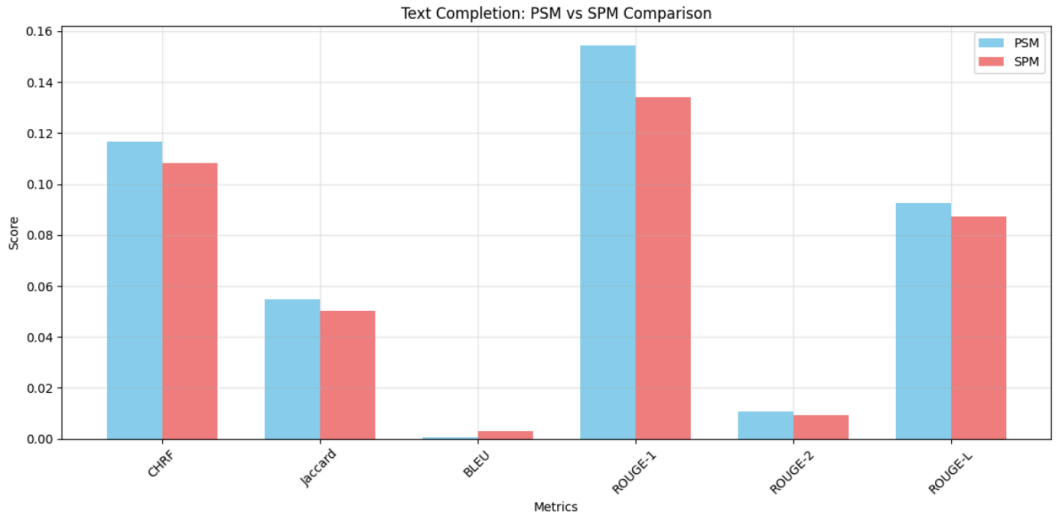
For every one of two datasets (one with code examples, one with text - Hamlet by William Shakespeare) I've created a SPM and PSM datasets.



Code Completion using PSM and SPM methods using code dataset

The empirical evaluation demonstrates that the Prefix-Suffix-Middle (PSM) prompting methodology consistently outperforms alternative approaches across all evaluated metrics in predicting the ground truth completion (MIDDLE segment). This superior performance is observed across multiple evaluation criteria, providing robust evidence for the effectiveness of the PSM approach in code completion tasks.

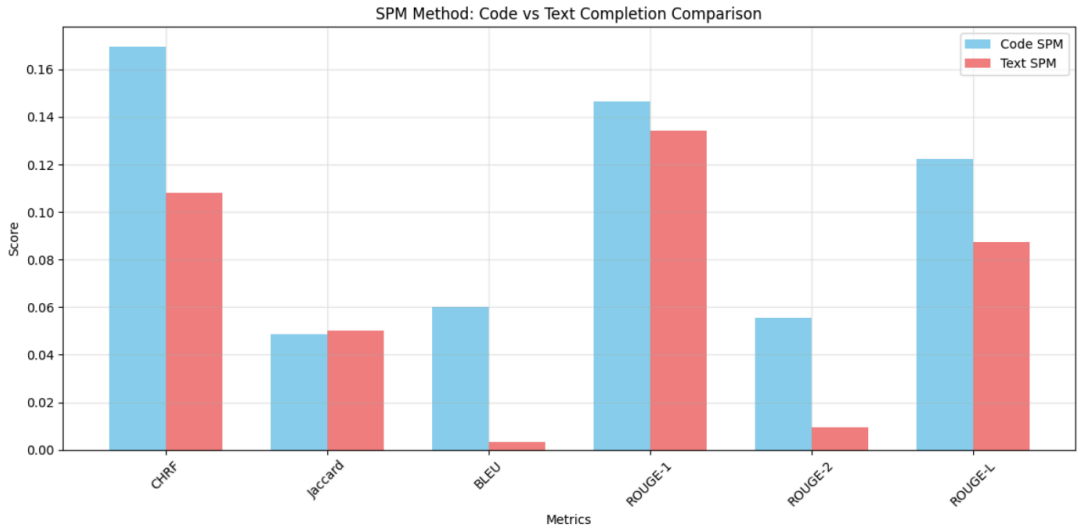
To validate the generalizability of these findings, I've extended the evaluation to natural language text datasets, yielding the following comparative results:



Code Completion using PSM and SPM methods using text dataset

The empirical results demonstrate that the Prefix-Suffix-Middle (PSM) method consistently outperforms the Suffix-Prefix-Middle (SPM) approach across evaluation metrics. This performance can be attributed to the model's enhanced ability to leverage contextual information when processing inputs beginning with the prefix. The prefix-first approach enables the model to establish a stronger semantic foundation before incorporating the suffix context, resulting in more coherent and accurate completions.

To evaluate the model's domain-specific capabilities, I've conducted a comparative analysis between code and natural language prediction tasks using Starcoder. While Starcoder was primarily optimized for code-related tasks, this comparison provides valuable insights into the model's cross-domain generalization capabilities and highlights the relative difficulty of natural language prediction compared to structured code completion tasks.



Code Completion using PSM and SPM methods using text dataset

The model's superior performance on code completion compared to text completion can be attributed to the fundamental differences between natural language and programming languages. For example, predicting "if (condition):" in Python has fewer valid continuations compared to predicting the next words in "The weather was..." where countless valid and contextually appropriate completions exist. This inherent difference in structural constraints makes code completion a more tractable problem for machine learning models.

The **Exact Match** metric in all examples scored 0, meaning we didn't find any exact matches between ground_truth and predicted middle sections. This provides us with insight that for large data, when performing ⅓ splits for MIDDLE, SUFFIX and PREFIX , it is almost impossible to reproduce the ground truth. The smaller the middle section to predict the more meaningful the Exact Match metric becomes.

When comparing other metrics I came to the conclusion that **ROUGE-1** may be the most insightful one for code prediction in our situation. It can capture partial matches even when the exact sequence isn't perfect, making it more suitable for large text evaluation where exact matches are unlikely.

6. Next steps

To enhance the Fill-in-Middle (FiM) code completion methodology, I would propose the following experimental investigations:

1) Segmentation Ratio Optimization:
    - Systematically vary the proportions of PREFIX, MIDDLE, and SUFFIX segments
    - Identify optimal ratios that balance computational efficiency with prediction accuracy
    - Evaluate performance across different segment size combinations (e.g., 40-20-40, 45-10-45)

2) Model Fine-tuning Strategy:
   Adjust key hyperparameters including:
    - Learning rate
    - Attention mechanisms
    - Context window size
    - Batch size

3) Experiment with different tokenization strategies

7. REFERENCES

[1] https://arxiv.org/pdf/2207.14255,Mohammad Bavarian , Heewoo Jun,Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, Mark Chen
Efficient Training of Language Models to Fill in the Middle, 2022

[2] https://arxiv.org/pdf/2309.06179v1, Shoutao Guo, Shaolei Zhang ,Yang Feng GLANCING FUTURE FOR SIMULTANEOUS MACHINE TRANSLATION, University of Chinese Academy of Sciences, Beijing, China 2023.

[3] https://en.wikipedia.org/wiki/Prefix_code,

[4] https://en.wikipedia.org/wiki/F-score,

[5] https://en.wikipedia.org/wiki/ROUGE_(metric),