Empowering Character-level Text Infilling by Eliminating Sub-Tokens

Anonymous ACL submission

Abstract

In infilling tasks, sub-tokens, representing instances where a complete token is segmented into two parts, often emerge at the boundaries of prefixes, middles, and suffixes. Traditional methods focused on training models at the token level, leading to sub-optimal performance in character-level infilling tasks during the inference stage. Alternately, some approaches considered character-level infilling but they relied on predicting sub-tokens in inference, yet this strategy diminished ability in character-level infilling tasks due to the large perplexity of the model on sub-tokens. In this paper, we introduce FIM-SE, which stands for Fill-In-the-Middle with both Starting and Ending character constraints. The proposed method addresses character-level infilling tasks by utilizing a line-level format to avoid predicting any sub-token in inference. In addition, we incorporate two special tokens to signify the rest incomplete lines, thereby enhancing generation guidance. Extensive experiments demonstrate that our proposed approach surpasses previous methods, offering a significant advantage. Code is available at https://anonymous.4open.science/r/FIM-SE-0678.

1 Introduction

The Transformer (Vaswani et al., 2017) decoder-only architecture has proven highly effective in various natural language processing (NLP) tasks. This success has paved the way for the development of advanced causal decoder-only models like GPT-4 (OpenAI, 2023), PaLM (Chowdhery et al., 2023; Anil et al., 2023), Llama (Touvron et al., 2023a,b; Rozière et al., 2023), and Falco (Penedo et al., 2023). These innovative models excel at generating coherent and contextually relevant responses to natural language prompts, showcasing state-of-the-art performance across various tasks, including question answering (Lewis et al., 2020b), logical reasoning (Kojima et al., 2022), and code synthesis (Li et al., 2023; Rozière et al., 2023).

Table 1: Examples for random splitting with Llama tokenizer, where the red, blue, and green text indicates the prefix, the middle, and the suffix, respectively. These four rows represent the pieces after randomly splitting, the sentence after exchanging suffix and middle, tokenized results, and token IDs, respectively.

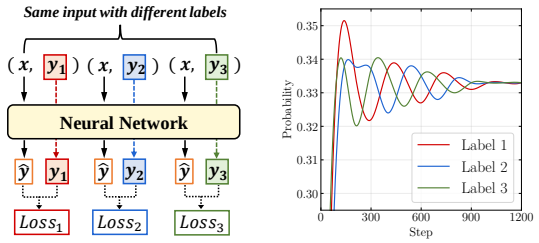
(a) The first splitting case.

Pieces	A fine day.
Reorder	A f day. ine
Tokens	[‘A’, ‘_f’, ‘day’, ‘.’, ‘ine’, ‘_’]
IDs	[29909, 285, 3250, 29889, 457, 29871]

(b) The second splitting case.

Pieces	A fine day.
Reorder	A fi day. ne
Tokens	[‘A’, ‘_fi’, ‘day’, ‘.’, ‘ne’, ‘_’]
IDs	[29909, 5713, 3250, 29889, 484, 29871]

Despite the success, the proficiency of these models is somewhat limited in tasks involving text infilling, which aims to generate text at a specific location within a prompt, while conditioning on both a prefix and a suffix. The main reason is their intrinsic left-to-right autoregressive design. To address this issue, CM3 (Aghajanyan et al., 2022) introduced the causal masking objective, placing a mask token at the intended fill location and completing the fill at the end. In contrast, FIM (Bavarian et al., 2022) proposed a fill-in-the-middle technique, which randomly divides documents into three segments and tags them with three special tokens. This technique then rearranges the middle and suffix segments, to use the prefix and the suffix to predict the middle segment in auto-regressive format. With these methods, decoder-only based models can effectively handle various infilling tasks and achieve excellent performance.



(a) Experiment. (b) Probabilities.

Figure 1: The probabilities of prediction when inconsistent labels appear in the training data.

However, employing the aforementioned methods may introduce inconsistencies during training. This arises from the potential division of a single token into multiple sub-tokens, as exemplified in Table 1. As we can see, due to character-level random splitting, the same prefixes ([29909]) have inconsistent objectives (285 and 5713, respectively) in different cases. The inconsistent objectives will significantly impact the model's perplexity, especially on sub-tokens. To illustrate, we construct a simple experiment on a classification task shown in Figure 1(a). The training data only contains three samples and they have the same input but different labels. We train a simple network on the training data and record the predicted probabilities of the three labels at each training step. As shown in Figure 1(b), the predicted probabilities for these three classes converge to 0.33, indicating a large perplexity of the model on the inconsistent objectives. The large perplexity on sub-tokens makes the probability of error increase when predicting a sub-token. This phenomenon is notably detrimental in sensitive tasks such as code completion, where even a minor error in any token can result in program malfunction. As a result, previous approaches have yet to fully inspire the potential of Transformer decoder-only models in infilling tasks.

To effectively address the issue, it is crucial to acknowledge and resolve an inherent conflict. (1) We need to avoid the model predicting sub-tokens. In the infilling training mode, the model's perplexity in sub-tokens is large, resulting in the low accuracy of predicting sub-tokens. (2) We need to output a sub-token when the user only writes part of a token. Because it is necessary to ensure that the output fits the context. If we directly drop several tokens to make sure no sub-tokens exist, the model's output may no longer align with the removed context, rendering it unreasonable in practical use.

Based on these concerns, we propose FIM-SE, which stands for Fill-In-the-Middle with both Starting and Ending character constraints. Our method enhances the organizational framework of FIM (Bavarian et al., 2022) to concurrently address the two scenarios mentioned earlier. In simple terms, we transfer the random-span infilling task to the multi-line infilling task. Specifically, after random character level splitting, we utilize two distinct special tokens to mark the *Last line of the Prefix (L-Prefix)* and the *First line of the Suffix (F-Suffix)*. The model is then tasked to generate text at line level that starts with *L-Prefix* and ends with *F-Suffix*. Their inclusion in the prompt simplifies the task for the model, facilitating the generation of text that seamlessly starts with *L-Prefix* and ends with *F-Suffix*. Overall, this method is designed to unlock the capabilities of decoder-only models in infilling tasks.

The core contribution of the paper is that we design a novel training method for the infilling task, a solution designed to effectively mitigate conflicts mentioned above in infilling tasks. Our method can effectively eliminate any potential inconsistencies and earnestly guarantee that the model's output aligns cohesively with the given context. Extensive experiments demonstrate the effectiveness of the proposed method on infilling tasks while not compromising code generation capabilities. Based on Code Llama 13B, our approach not only achieves an 8.8% enhancement in the Humaneval random-span infilling task, with substantial improvements of 11.5% and 10.7% in the single-line and multi-line infilling tasks respectively, but also maintains minimal impact on the model's performance in code generation tasks.

2 Related Work

2.1 Large Language Models for Infilling

Various Large Language Models (LLMs) have been developed for general generation tasks (Touvron et al., 2023a; Xiong et al., 2023; Chowdhery et al., 2023; Penedo et al., 2023; Jiang et al., 2023; Yang et al., 2023; Bai et al., 2023). Most of these models adopt a left-to-right autoregressive generation approach due to its effectiveness, as validated by research such as GPT (Radford et al., 2018, 2019; Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023). In the realm of code-related tasks, where infilling is essential, LLMs are specifically trained for this task. For instance, Incoder (Fried et al.,

2023) utilizes a causal masking objective, while SantaCoder (Allal et al., 2023), Starcoder (Li et al., 2023), and Code Llama (Rozière et al., 2023) employ the fill-in-the-middle technique introduced by FIM (Bavarian et al., 2022).

2.2 Text Infilling Models

The infilling task plays a crucial role in numerous real-world applications, including document editing[1] and code completion[2]. Three common Transformer (Vaswani et al., 2017) architectures are capable of executing this task, *i.e.,* encoder-only, encoder-decoder, decoder-only. In encoder-only architectures, masked language modeling is employed as the pre-training task, exemplified by models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). These models are designed to infill brief spans, ranging from a single token (Devlin et al., 2019) to a word (Cui et al., 2021), and even several contiguous tokens (Joshi et al., 2020). In encoder-decoder architectures, a common approach involves masking several tokens in the encoder and then tasking the model with decoding the complete sentence, as exemplified by MASS (Song et al., 2019). Additionally, models like Bart (Lewis et al., 2020a) and T5 (Raffel et al., 2020) have introduced an infilling noising method. This technique replaces multiple tokens with a single mask token, challenging the model to decode the masked span. In decoder-only architectures, several methods are employed for infilling tasks. The Insertion Transformer (Stern et al., 2019) instructs the model to first determine the location for the next token, followed by the token prediction itself. Meanwhile, GLM (Du et al., 2022), CM3 (Aghajanyan et al., 2022), and Incoder (Fried et al., 2023) adopt a different approach. They shift the target span to the end of the context, employing left-to-right autoregressive modeling for training.

Most of these models are designed for token-level infilling tasks, which often don't align with real-world applications due to the incomplete nature of the final token in actual prompts. FIM (Bavarian et al., 2022) explored various levels of spans, *i.e.,* line level, token level, and character level. As shown in their results, models trained with line-level or token-level spans perform poorly on character-level infilling tasks. To enhance the performance of models trained on token-level spans in character-level infilling tasks, token healing was proposed to fix tokenization artifacts that normally arise at the boundary between the end of a prompt and the beginning of a set of generated tokens[3]. While it effectively bridges the gap between the prefix and generated text, it falls short in handling the transition between generated text and the suffix, highlighting the need for further research in character-level infilling.

3 Preliminaries

In this section, we provide a straightforward introduction to the FIM (Bavarian et al., 2022) method and conduct a theoretical analysis of how inconsistent labeling affects the model's perplexity.

3.1 Fill-In-the-Middle (FIM)

FIM is designed to train models to complete the central sections of documents. This approach involves joint training on a combined dataset of traditional left-to-right sequences and data transformed by FIM, with an infilling rate reaching as high as 90%. According to experimental results, FIM maintains the autoregressive test losses of the left-to-right models without incurring significant costs, and it only slightly impacts the performance in downstream evaluations (Allal et al., 2023).

In a particular document, FIM segments a document into three distinct parts: the prefix, the middle, and the suffix. It introduces three levels of segmentation: single-line, multi-line, and random-span. Because random-span is more in line with actual usage conditions, previous studies (Rozière et al., 2023; Li et al., 2023) usually trained the model with random-span level splitting. After splitting, it moves the middle piece to the end as

$$\text{doc} \rightarrow (\text{pre}, \text{mid}, \text{suf}) \rightarrow (\text{pre}, \text{suf}, \text{mid}),$$

then concatenate the three pieces using special tokens as

<PRE> pre <SUF> suf <MID> mid <EOT>.

This mode is termed Prefix-Suffix-Middle (PSM) mode. Additionally, FIM introduced the Suffix-Prefix-Middle (SPM) mode, which interchanges the positions of the prefix and suffix. A variant of the SPM mode is also proposed, maintaining the same structure as the PSM mode. Detailed descriptions of these modes are provided in Appendix A.

[1] https://copilot.microsoft.com
[2] https://github.com/features/copilot
[3] https://github.com/guidance-ai/guidance/blob/main/notebooks/tutorials/token_healing.ipynb

3.2 Impact of Inconsistent Labels

When the FIM method employs the random-span approach, a training sample can contain up to four sub-tokens. This can potentially lead to inconsistent labels that indicate the same input but with different labels. This issue becomes particularly critical when the model is required to predict a sub-token. In Section 1, we construct a simple experiment to illustrate that this inconsistency can significantly affect the model's perplexity. Here, we offer a theoretical analysis to further elucidate this phenomenon.

We are considering a classification task involving n classes, where each sample's label is associated with one of m different categories across various training instances. Let $\boldsymbol{y} \in \{0,1\}^n$ represent the actual label, and $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ represent the predicted probabilities. In this context, the cross-entropy loss is computed as

$$\mathcal{L} = -\sum_{i=0}^{n} \boldsymbol{y}_i \log \hat{\boldsymbol{y}}_i. \tag{1}$$

Assuming that the first m elements of $\boldsymbol{y}$ (i.e., $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$) are set to 1, while the rest are 0, the loss function is defined as

$$\mathcal{L}(\hat{\boldsymbol{y}}) = -\sum_{i=0}^{m} \log \hat{\boldsymbol{y}}_i. \tag{2}$$

Then our objective can be expressed as

$$\hat{\boldsymbol{y}}^* = \arg\max \mathcal{L}(\hat{\boldsymbol{y}}), \quad s.t. \sum_{i=0}^{n} \hat{\boldsymbol{y}}_i = 1. \tag{3}$$

Here, we introduce the concept of the Lagrange Multiplier, which enables us to formulate the target function as

$$\mathcal{L}(\hat{\boldsymbol{y}}, \lambda) = -\sum_{i=0}^{m} \log \hat{\boldsymbol{y}}_i - \lambda(\sum_{i=0}^{n} \hat{\boldsymbol{y}}_i - 1). \tag{4}$$

We then calculate the partial derivatives of $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_m$ respectively, which allows us to obtain

$$\frac{\partial \mathcal{L}(\hat{\boldsymbol{y}}, \lambda)}{\partial \hat{\boldsymbol{y}}_i} = -\frac{1}{\hat{\boldsymbol{y}}_i} - \lambda \quad \text{when} \quad i = 1, \ldots, m. \tag{5}$$

By setting these derivatives to zero, we obtain

$$\hat{\boldsymbol{y}}_1^* = \cdots = \hat{\boldsymbol{y}}_m^* = -\frac{1}{\lambda}. \tag{6}$$

Since $\hat{\boldsymbol{y}}_{m+1}, \ldots, \hat{\boldsymbol{y}}_n$ are not included in the objective function of Eq. (3), and given that the logarithm is a monotonically increasing function, setting these values to 0 would maximize the objective function. Consequently, the condition is formulated as $\sum_{i=1}^{m} \hat{\boldsymbol{y}}_i = 1$. By incorporating this condition, we derive

$$\hat{\boldsymbol{y}}_1^* = \cdots = \hat{\boldsymbol{y}}_m^* = \frac{1}{m}. \tag{7}$$

We have now completed the proof, demonstrating that when a data point is labeled differently across various samples, the model tends to assign an equal probability of $\frac{1}{m}$ to each label. This behavior leads to a large perplexity of the model, which further suggests its limited modeling capability.

4 Methodology

In this section, we introduce the proposed method. We begin by outlining the training process with FIM-SE, followed by an explanation of the inference procedure. Finally, we delve into more training details and highlight the distinctions between our approach and the traditional FIM method.

4.1 FIM-SE Training

The core idea of FIM-SE is to ensure that the tokens predicted by the model are complete, thereby circumventing the issue of large perplexity associated with sub-tokens. Specifically, we shift from character-level to line-level random splitting in training data construction and then reconstruct the prompt to keep the ability of the model on the character-level infilling tasks.

As shown in Figure 2, our process for forming the final training sample from a specific document involves three distinct steps. (1) Splitting: we split the original document into three pieces at the character level, namely the prefix, the middle, and the suffix (2) Refining: we distinguish between the last line of the prefix and the first line of the suffix, denoting them as *L-Prefix* and *F-Suffix*, respectively. Correspondingly, we label the remaining lines of the prefix and the suffix as *R-Prefix* and *R-Suffix*. (3) Concatenating: we concatenate all these sections in the following order along with their special tokens as

<PRE> *R-Prefix* <SUF> *R-Suffix*
<START> *L-Prefix* <END> *F-Suffix*
<MID> *L-Prefix Middle F-Suffix* <EOT>.

When tokenizing a sample, we tokenize each section individually and then concatenate them with the special tokens, which ensures that special tokens will not be cut or merged.

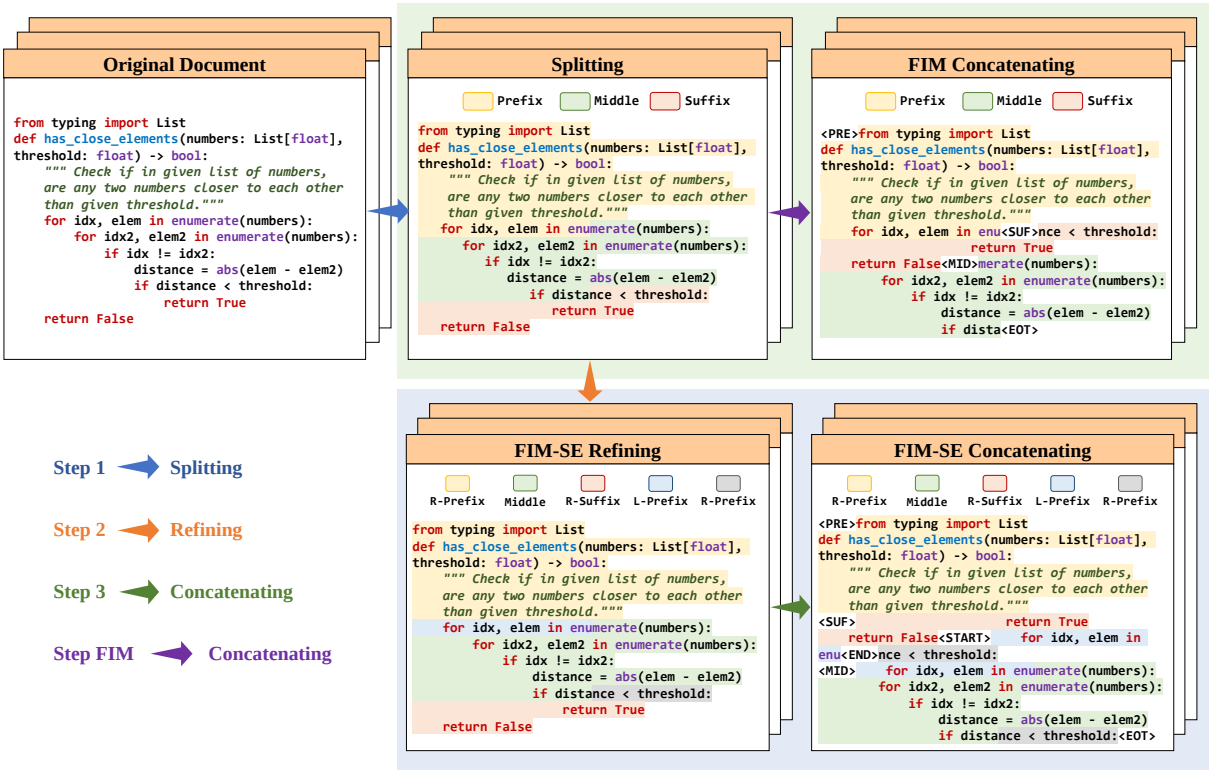


Figure 2: An overview of the difference between FIM and the proposed FIM-SE. Here, the green background indicates vanilla FIM and the blue background indicates our FIM-SE.

4.2 FIM-SE Inference

During the inference stage, the model can be employed for left-to-right generation in a standard manner. When working with an arbitrary location within an existing document, we establish the preceding lines as *R-Prefix* and the following lines as *R-Suffix*. For the specific line at the target location, the text before this point is termed *L-Prefix*, and the text following it is named *F-Suffix*. Subsequently, a span is generated to be inserted at this location by autoregressively sampling tokens from the structured prompt

<PRE> *R-Prefix* <SUF> *R-Suffix*
<START> *L-Prefix* <END> *F-Suffix*<MID>.

This process continues until the "<EOT>" (End of Text) token is produced.

After obtaining the generation, we verify if it begins with the *L-Prefix* and ends with the *F-Suffix*. If the generation does not adhere to these criteria, we classify the infilling endeavor as unsuccessful. Conversely, if the criteria are met, we eliminate the *L-Prefix* from the beginning and the *F-Suffix* from the end, considering the remaining text as the completed segment.

4.3 Learning and Discussion

Training Details. We train our models using the Starcoder code corpus[4], a carefully curated dataset sourced from The Stack, encompassing 92 languages. To ensure consistency, we exclude categories such as GitHub issues, GitHub commits, and Jupyter Notebooks, which possess distinct column structures. Additionally, we remove flags marking repositories, files, and stars to maintain a focus on the pure code content in the remaining files. After gathering the data, we process it using the previously described method with a 90% FIM rate, following the methodologies of existing studies (Bavarian et al., 2022; Allal et al., 2023; Rozière et al., 2023). It's important to note that we exclusively employ the PSM format depicted in Figure 2, as the SPM variant used in prior research (Bavarian et al., 2022) lacks a separator between the prefix and middle, potentially leading to model confusion. We conduct experiments and give an experimental analysis in Section 5.3.

Discussion. Compared to previous masked language modeling on encoder-only models and

[4] https://huggingface.co/datasets/bigcode/starcoderdata

encoder-decoder models, our method excels in character-level infilling. While these traditional methods primarily concentrate on token-level infilling, this approach often falls short in numerous industry applications, as user text seldom forms complete tokens. Compared to vanilla FIM (Bavarian et al., 2022), our method also has the following two merits. Firstly, our method ensures that tokens following "<MID>" are complete, eliminating the need for sub-token predictions during inference and thereby mitigating the effects of the large perplexity of the model on sub-tokens. Secondly, our method transforms character-level infilling into line-level infilling. This unification of formats enhances transfer across different levels, significantly augmenting the efficacy of FIM training.

5 Experiments

In this section, we construct experiments to demonstrate the effectiveness of our method. Due to space limitations, we have constructed more experiments in Appendix B.

5.1 Experimental Setup

Datasets. Following FIM (Bavarian et al., 2022), we use code to test our methods. Because we can use test suites to evaluate the correctness of samples in our tasks even when evaluating long samples from open-ended generations. Specifically, we use three levels of infilling benchmarks, namely random-span, single-line, and multi-line. All of them are constructed from Humaneval benchmarks (Chen et al., 2021). Since other infilling benchmarks such as Return Type Prediction and Docstring Generation focus on token-level infilling, we do not use these benchmarks.

Implementation Details. We continually pre-train four models with our methods, *i.e.,* Starcoder-1B, Starcoder-15B (Li et al., 2023), Code Llama 7B, and Code Llama 13B (Rozière et al., 2023). We employ AdamW (Loshchilov and Hutter, 2019) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$ and weight decay of 0.1. Following previous study (Gupta et al., 2023), we set the peak learning rate to 3×10^{-5} and use a cosine schedule without warm-up. We use a batch size of 4M tokens which are presented as sequences of 8K tokens each for Starcoder and 16K tokens each for Code Llama. We train each model on 20B tokens in total. To efficiently train the computationally intensive models, we simultaneously employ DeepSpeed (Rajbhandari et al., 2020) and Flash Attention 2 (Dao, 2023). On 32 NVIDIA A800 80GB GPUs, Starcoder-1B, Starcoder-15B, Code Llama 7B, and Code Llama 13B take 14 hours, 140 hours, 75 hours, and 138 hours, respectively.

5.2 Results

Baselines. We compare FIM-SE with previous state-of-the-art methods, including Incoder (Fried et al., 2023), FIM (Bavarian et al., 2022), Codex (Chen et al., 2021), Starcoder (Li et al., 2023) and Code Llama (Rozière et al., 2023). For other code models such as CodeGeeX (Zheng et al., 2023) and OctoCoder (Muennighoff et al., 2023), we do not use them as baselines since they have not undergone infilling pre-training.

Random-span. As shown in Table 2, our proposed method demonstrates notable improvements in random-span infilling tasks across four models, specifically achieving gains of 4.7%, 1.3%, 8.1%, and 8.8%. Notably, the enhancement in Starcoder-15B is comparatively modest. This could be attributed to the fact that Starcoder has undergone pre-training with four epochs on a total of 1TB tokens, in contrast to Code Llama's pre-training on 500B tokens, resulting in a more refined model fit. Comparing Starcoder-15B with Starcoder-1B, the small model trained on the same tokens has more gain, suggesting that the consistent training approach of our method is particularly beneficial for smaller models in achieving better fit. Comparing Starcoder-15B with Code Llama 13B, the model with a similar size using fewer tokens achieves better results. This indicates that the consistent training approach of our method accelerates the fitting process in larger models.

Single-line and Multi-line. The proposed method demonstrates notable improvements in both single-line and multi-line infilling tasks. For instance, based on Code Llama 13B, our method surpasses FIM by 11.5% and 10.7% in single-line and multi-line infilling tasks, respectively. This enhancement can be attributed to two key factors. Firstly, our method integrates character-level and line-level processing, significantly enhancing the model's line-level infilling capabilities. Secondly, it avoids the inclusion of any sub-tokens after the "<MID>" token, which sharpens the model's accuracy in predicting the initial token. In contrast, in the standard FIM, the first token following "<MID>" is typically a sub-token during training, while the

Table 2: Comparison results on Humaneval infilling datasets. Results evaluated on our end are marked with "*", while those unavailable are left blank. Note that StarCoder was evaluated using a cleaned and smaller version of MBPP so we conducted a re-evaluation.

Model	Size	Training Methods	random-span	single-line	multi-line	Humaneval	MBPP
Incoder	6B	Causal Masking	-	69.0	38.6	15.0	19.4
FIM	7B	FIM-SPM	55.1	75.1	44.1	-	-
code-davinci-002	175B	-	74.2	91.6	69.9	44.5	55.4
Starcoder	1B	FIM-PSM	44.1*	64.3*	30.8*	15.2	22.6*
		FIM-SE-PSM	**48.8 (+4.7)**	**72.6 (+8.3)**	**37.1 (+6.3)**	**16.5**	**25.6**
	15B	FIM-PSM	66.4*	83.8*	53.7*	30.4	43.2*
		FIM-SE-PSM	**67.7 (+1.3)**	**85.8 (+2.0)**	**57.4 (+3.7)**	**30.5**	**44.6**
Code Llama	7B	FIM-SPM	39.0	83.3	50.8	**33.5**	**41.4**
		FIM-PSM	59.7	74.1	48.2		
		FIM-SE-PSM	**67.8 (+8.1)**	**84.9 (+10.8)**	**57.2 (+9.0)**	30.5	**41.4**
	13B	FIM-SPM	41.9	85.6	56.1	36.0	47.0
		FIM-PSM	63.6	75.9	51.0		
		FIM-SE-PSM	**72.4 (+8.8)**	**87.4 (+11.5)**	**61.7 (+10.7)**	**37.2**	**50.2**

model is adopted to predict a complete token in the line-level infilling tasks during the inference stage. A comprehensive case study is provided in Appendix B.2 for further illustration.

Code Generation Task. We also report results on Humaneval (Chen et al., 2021) and MBPP (Austin et al., 2021). As shown in Table 2, our method has minimal impact on the model's performance in the two code generation tasks (Note that we cannot reproduce the result of Code Llama 7B on Humaneval, just 29.9% in our environment). In summary, FIM-SE demonstrates a remarkable ability to improve infilling tasks without compromising code generation capabilities.

5.3 Detail Analysis

Impact of Inconsistent Labels. As mentioned in Section 1 and Section 3.2, we discussed how FIM leads to inconsistent labels during training at split points. This phenomenon results in large perplexity on sub-tokens, subsequently diminishing the model's accuracy in generating sub-tokens. To investigate this effect, we conducted an experiment based on the Starcoder-1B. Specifically, we adjusted the temperature within the range of [0, 1.4] and compared the performance of models trained using both FIM-SE and FIM in generating 20 completions to estimate the Pass@1 rate. Figure 3 illustrates that the performance gap between the FIM-SE and FIM generators widens as the temperature increases, highlighting the larger perplexity associated with models trained using FIM.

Furthermore, we evaluate the impact of inconsistent labels on training. Specifically, we mask the loss for tokens in *L-Prefix* and *F-Suffix*, ensuring that only complete tokens contribute to loss calculations. As shown in table 3, computing losses for *L-Prefix* and *F-Suffix* led to a slightly higher test loss without significantly affecting performance. This could be attributed to the minimal proportion of sub-tokens, as the presence of up to four sub-tokens per sample had a negligible impact on the final test results. In summary, while the loss of sub-tokens in training scarcely affects performance,

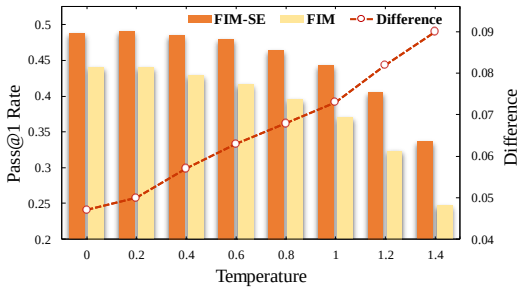


Figure 3: Performance on Humaneval random-span infilling task with different temperatures. The line denotes the difference between FIM-SE and FIM. Note that when the temperature surpasses 1.4, both models output noisy text and show very low performance.

Table 3: Effect of training loss on sub-tokens. Here, LF-Loss denotes the loss for tokens in *L-Prefix* and *F-Suffix*.

Methods	FIM-SE	w/o LF-Loss
random-span	**0.488**	**0.488**
single-line	**0.726**	0.716
multi-line	**0.371**	0.369
Test loss	0.847	**0.834**

Table 4: Comparison between different SPM format variants and FIM-SE.

Methods	random-span	single-line	multi-line
FIM-SE	0.488	**0.726**	0.371
SPM v1	**0.492**	0.703	0.374
SPM v2	0.013	0.085	0.088
SPM v3	0.090	0.717	**0.383**

the presence of sub-tokens in prediction objectives markedly influences performance.

Comparison with SPM Mode. In previous studies, the Suffix-Prefix-Middle variant had better performance in most cases (Bavarian et al., 2022; Rozière et al., 2023). Here, we explore how to combine our method with SPM mode based on Starcoder-1B. Specifically, we designed the following three prompt formats for SPM mode. We train the model using these formats and the PSM mode, equally distributed across 20 billion tokens.

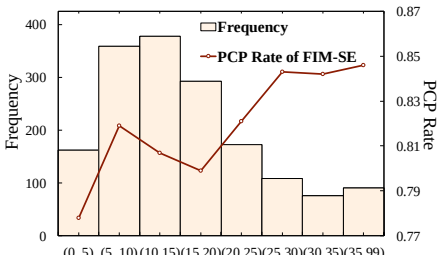
(1) SPM v1: "<SUF> *R-Suffix* <PRE> *R-Prefix* <START> *L-Prefix* <END> *F-Suffix* <MID>", which add the constraints before the middle to the vanilla SPM mode.

(2) SPM v2: "<PRE> <SUF> *R-Suffix* <START> *L-Prefix* <END> *F-Suffix* <MID> *R-Prefix*", which add the constraints before the middle to the variant SPM in FIM.

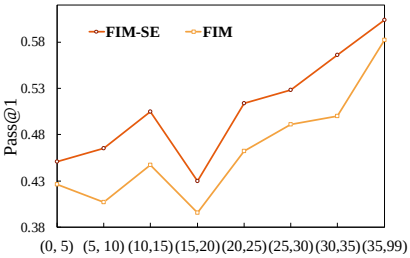
(3) SPM v3: "<PRE> <SUF> *R-Suffix* <MID> *R-Prefix* <START> *L-Prefix* <END> *F-Suffix*", which add the constraints after prefix to the variant SPM in FIM.

Table 4 presents all comparison results of the three variants. As we can see, SPM v2 and SPM v3 perform worse on random-span infilling tasks. This occurs because there is no separator between the prefix and the middle, leading to conflicts with the PSM mode, regardless of where the restriction is inserted. In contrast, SPM v1 and PSM perform almost the same because there is no conflict. To maintain consistency with the pre-trained models (Li et al., 2023; Rozière et al., 2023), we adopt the PSM mode.

Analysis of Post-Check during Inference. As we mentioned in Section 4.2, it's essential to verify if the generation begins with the *L-Prefix* and ends with the *F-Suffix*. Here, we perform statistical analysis on the success rate of the model based on Starcoder-1B. We focus on the Post-Check Pass Rate (PCP Rate), which quantifies the percentage



(a) PCP Rate of FIM-SE.



(b) Pass@1 performance of FIM-SE and FIM.

Figure 4: Statistics of length of *L-Prefix* and *F-Suffix*.

of model outputs complying with the post-check criteria, *i.e.,* starting with the *L-Prefix* and ending with the *F-Suffix*. We then examine the correlation between the PCP Rate and the average length of the *L-Prefix* and *F-Suffix*. Additionally, we analyze the Pass@1 rates for FIM-SE and FIM across varying lengths of these prefixes and suffixes.

As shown in Figure 4, the PCP Rate increases with length, suggesting that longer *L-Prefixes* and *F-Suffixes* provide more guidance for the model's text completion. Moreover, the Pass@1 metrics for both FIM-SE and FIM also support this, showing enhanced performance with extended *L-Prefixes* and *F-Suffixes*. Across all lengths, FIM-SE consistently outperforms the standard FIM, demonstrating the effectiveness of our approach.

6 Conclusion

In this paper, we showed that traditional infilling techniques struggle with managing the boundaries of prefixes and suffixes. To address this, we introduced a novel approach, referred to as FIM-SE. Our method transforms the random-span mode to multi-line mode by removing the *L-Prefix* and *F-Suffix*. We further incorporated two special tokens to delineate the two incomplete lines, thereby guiding the generation. Extensive experiments reveal that our approach surpasses existing baselines with a clear edge. In future work, we plan to explore the adaptation of our method to the variant SPM mode, which holds the promise of even better performance.

7 Limitations

The primary limitation of this study is the inability of the proposed method to accommodate the variant SPM mode, previously established as superior by prior research. This challenge arises due to the absence of a distinct delimiter between the prefix and middle, impeding our capacity to guide the model on the commencement point for completion and to appropriately position the prompt that instructs the model to start with *L-Prefix* and end with *F-Suffix*. In future endeavors, we plan to explore adapting our method to the variant SPM mode, to achieve better performance. Another limitation of this paper is the probability that our proposed method fails to complete tasks when the generation neither starts with the *L-Prefix* nor ends with the *F-Suffix*. For example, the fail rates of Starcoder-1B and Starcoder-15B are 18.7% and 9.4%, respectively. This issue is a primary factor impacting model performance. Future work will concentrate on improving the post-check pass rate by developing more comprehensive prompts and refining constraint decoding.

8 Ethics Statement

In this paper, we utilized the Starcoder dataset (Li et al., 2023). This dataset has been made publicly available for academic purposes. The creators of the Starcoder dataset have transparently disclosed its derivation from The Stack v1.2 (Kocetkov et al., 2022). Importantly, The Stack v1.2 is compiled from a collection of GitHub repositories, all of which operate under permissive licenses. This ensures that the data's utilization aligns with the original authors' intentions and the legal frameworks governing open-source contributions. In conclusion, the application of the Starcoder dataset in our study complies with the ethical guidelines for research data usage, aligning with the broader principles of academic honesty and the responsible conduct of research.

References

Armen Aghajanyan, Bernie Huang, Candace Ross, Vladimir Karpukhin, Hu Xu, Naman Goyal, Dmytro Okhonko, Mandar Joshi, Gargi Ghosh, Mike Lewis, and Luke Zettlemoyer. 2022. CM3: A causal masked multimodal model of the internet. *CoRR*, abs/2201.07520.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Muñoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, Logesh Kumar Umapathi, Carolyn Jane Anderson, Yangtian Zi, Joel Lamy-Poirier, Hailey Schoelkopf, Sergey Troshin, Dmitry Abulkhanov, Manuel Romero, Michael Lappert, Francesco De Toni, Bernardo García del Río, Qian Liu, Shamik Bose, Urvashi Bhattacharyya, Terry Yue Zhuo, Ian Yu, Paulo Villegas, Marco Zocca, Sourab Mangrulkar, David Lansky, Huu Nguyen, Danish Contractor, Luis Villa, Jia Li, Dzmitry Bahdanau, Yacine Jernite, Sean Hughes, Daniel Fried, Arjun Guha, Harm de Vries, and Leandro von Werra. 2023. Santacoder: don't reach for the stars! *CoRR*, abs/2301.03988.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. 2023. Palm 2 technical report. *CoRR*, abs/2305.10403.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *CoRR*, abs/2309.16609.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient training of language models to fill in the middle. *CoRR*, abs/2207.14255.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child,

Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, and Ziqing Yang. 2021. Pre-training with whole word masking for chinese BERT. *IEEE ACM Trans. Audio Speech Lang. Process.*, 29:3504–3514.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *CoRR*, abs/2307.08691.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: general language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 320–335. Association for Computational Linguistics.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. 2023. Continual pre-training of large language models: How to (re)warm your model? *CoRR*, abs/2308.04014.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. *Trans. Assoc. Comput. Linguistics*, 8:64–77.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 TB of permissively licensed source code. *CoRR*, abs/2211.15533.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing*

Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 66–71. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020b. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you! *CoRR*, abs/2305.06161.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. 2023. Octopack: Instruction tuning code large language models. *CoRR*, abs/2308.07124.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon LLM: outperforming curated corpora with web data, and web data only. *CoRR*, abs/2306.01116.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. *OpenAI*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: masked sequence to sequence pre-training for language generation. In *Proceedings*

of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936. PMLR.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985. PMLR.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/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. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. 2023. Effective long-context scaling of foundation models. *CoRR*, abs/2309.16039.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, Fan Yang, Fei Deng, Feng Wang, Feng Liu, Guangwei Ai, Guosheng Dong, Haizhou Zhao, Hang Xu, Haoze Sun, Hongda Zhang, Hui Liu, Jiaming Ji, Jian Xie, Juntao Dai, Kun Fang, Lei Su, Liang Song, Lifeng Liu, Liyun Ru, Luyao Ma, Mang Wang, Mickel Liu, MingAn Lin, Nuolan Nie, Peidong Guo, Ruiyang Sun, Tao Zhang, Tianpeng Li, Tianyu Li, Wei Cheng, Weipeng Chen, Xiangrong Zeng, Xiaochuan Wang, Xiaoxi Chen, Xin Men, Xin Yu, Xuehai Pan, Yanjun Shen, Yiding Wang, Yiyu Li, Youxin Jiang, Yuchen Gao, Yupeng Zhang, Zenan Zhou, and Zhiying Wu. 2023. Baichuan 2: Open large-scale language models. *CoRR*, abs/2309.10305.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5673–5684. ACM.

Appendix

A Fill-In-the-Middle (FIM)

In Section 3.1, We briefly introduced the prefix-suffix-middle (PSM) mode of FIM (Bavarian et al., 2022). Here, we give a detailed description of the suffix-prefix-middle (SPM) mode and a variant SPM mode.

For the vanilla SPM mode, it just swaps the prefix and the suffix. Specifically, after splitting, it moves the suffix to the before:

$$\text{doc} \rightarrow (\text{pre, mid, suf}) \rightarrow (\text{suf, pre, mid}),$$

then concatenate the three pieces using special tokens as

<SUF> suf <PRE> pre <MID> mid <EOT>.

To maximize transfer between PSM mode and SPM mode, FIM proposed a novel variant of SPM mode, which concatenates the prefix, the middle, and the suffix pieces as

<PRE> <SUF> suf <MID> pre mid <EOT>.

The format occurs naturally as part of PSM training when the chosen prefix is empty. In this way, the two modes have a consistent format and they can transfer with each other in joint training and maximize the profits.

B Additional Experiments

B.1 Comparison with Token Healing

As discussed in Section 2, token healing is proposed as an ideal solution for addressing tokenization that normally arise at the boundary between the end of a prompt and the beginning of a set of generated tokens. Here, we evaluate our approach against the token healing method based on Starcoder-1B. However, since token healing struggles with the split points at the end of generated tokens and the subsequent suffix, we integrate it with our method for a comprehensive solution. Specifically, we construct the prompt as "<PRE> *R-Prefix* <SUF> *R-Suffix* <START> *L-Prefix* <END> *F-Suffix* <MID> *L-Prefix*" and focus solely on verifying if the generated text ends with *F-Suffix*.

Table 5 presents the comparison results between our method and token healing. Surprisingly, token healing performs slightly worse than our method. Detailed analysis revealed that token healing struggles with complex scenarios, such as splitting the last token into two sub-tokens and merging the latter sub-token with the initial generated token.

Here, we provide an in-depth analysis of the situation. Token healing backs up the generation process by one or more tokens before the end of the prompt, then constrains the first tokens generated to have a prefix that matches the last token in the prompt. As illustrated in Table 6(a), if the last token in the prompt is "so," token healing identifies a token that both matches this last token's prefix and possesses the highest probability, such as "sort." Consequently, the first generated token is seamlessly integrated, allowing the generation process to proceed smoothly.

However, due to the consistent integrated intrinsic features of the sentencepiece (Kudo and Richardson, 2018) algorithm, it tends to merge the last sub-token with the preceding one if possible. For example, as illustrated in Table 6(b), the word "delimiter" is tokenized into "deli" and "meter." If a prompt ends with "delim", the algorithm prefers tokenizing this as "delim" instead of splitting it into "deli" and "m". Token healing does not intervene, because there is no token starting with "delim". Consequently, when the last sub-token can be com-

Table 5: Comparison with Token Healing.

Methods	random-span
FIM-SE	0.488
Token Healing	0.484

Table 6: Case of token healing. The first case can be perfectly solved by token healing. The second case cannot be solved by token healing. Here, '_' denotes blank.

(a) A case can be solved by token healing.

Piece	Raw Text	Tokens				
Prefix	def so	def	_ so			
Output	def sort(arr)	def	_ sort	(	arr	)
Label	def sort(arr)	def	_ sort	(	arr	)

(b) A case cannot be solved by token healing.

Piece	Raw Text	Tokens			
Prefix	r.add(delim	r delim	.	add	(
Output	r.add(delimter)	r delim	. ter	add)	(
Label	r.add(delimeter)	r deli	. meter	add)	(

Prefix

```
def largest_prime_factor(n: int):
    """Return the largest prime factor of n. Assume n > 1 and is not a prime.
    >>> largest_prime_factor(13195)
    29
    """
    def is_prime(k):
        if k < 2:
            return False
        for i in range(2, k - 1):
            if k % i == 0:
                return False
        return True
```

Suffix

```
    for j in range(2, n + 1):
        if n % j == 0 and is_prime(j):
            largest = max(largest, j)
    return largest
```

Target Middle

```
    largest = 1
```

The top five choices for the initial generated token on Starcoder-1B (FIM), along with their probabilities

'\n': 0.463; '\n_ _ _ ': 0.225; '_ _ _ _': 0.073; '<|endoftext|>': 0.068; '_ _ _ _\n_ _ _ ': 0.061;

The top five choices for the initial generated token on Starcoder-1B (FIM-SE), along with their probabilities

'_ _ _ ': 0.829; '\n_ _ _ ': 0.115; '_ _ _ _\n_ _ _ ': 0.037; '_ _ _ _': 0.008; '_ _ _ _ _ _ _ _\n_ _ _ ' : 0.002;

Table 7: A Case to show perplexity of models on the initial generated token. Here, '_' denotes blank, and '\n' denotes newline.

bined with the previous one, token healing is unable to rectify it effectively.

To effectively resolve this issue, it is necessary to revert several tokens and subsequently engage in limited decoding, utilizing a Trie tree, until the regenerated text encompasses the previously rolled-back tokens. Nonetheless, this approach is time-intensive as each decoding step requires traversing the Trie tree to identify all tokens corresponding to the given prefix. In contrast, our method only requires modifying the prompt and doing some post-processing. In addition, our method can handle both boundaries between the prefix and middle as well as boundaries between the suffix and middle.

B.2 Case Study

Here, we present a case demonstrating the model's large perplexity on the initial generated token based on Starcoder-1B. Table 7 illustrates that, despite being a single-line infilling scenario, the perplexity for the first token is remarkably large, significantly influencing the overall generation. This is primarily because the first token following "<MID>" tends to be a sub-token during training, varying across samples due to random character-level splitting. In contrast, our approach guarantees that no sub-token prediction is required, leading to lower perplexity and enhanced performance.

Based on these concerns, we hypothesize that the superior performance of the variant SPM mode over the PSM mode, particularly evident in the single-line infilling task on Code Llama (Rozière et al., 2023) (85.6% vs. 75.9%), can be attributed to the specific processing format employed by Code Llama. In the format "<PRE> <SUF> suf <MID> pre mid <EOT>", Code Llama initially merges the prefix and middle segments before tokenization. This approach ensures that, following the "<MID>" token, there are no sub-tokens except for the initial token. Consequently, this format also guarantees that no sub-token prediction is required when the prefix is not empty, contributing to the enhanced performance of the variant SPM mode. In contrast, FIM (Bavarian et al., 2022) adopts a different approach by tokenizing the prefix and the middle separately before concatenating them. This leads to the presence of sub-tokens amidst the tokens. Consequently, the performance gap between SPM and PSM modes in FIM is narrower (61.6% vs. 62.2%) compared to that in Code Llama.