
Efficient Training of Language Models to Fill in the Middle

Mohammad Bavarian *

Heewoo Jun *

Nikolas Tezak

John Schulman

Christine McLeavey

Jerry Tworek

Mark Chen

OpenAI

Abstract

We show that autoregressive language models can learn to infill text after we apply a straightforward transformation to the dataset, which simply moves a span of text from the middle of a document to its end. While this data augmentation has garnered much interest in recent years, we provide extensive evidence that training models with a large fraction of data transformed in this way does not harm the original left-to-right generative capability, as measured by perplexity and sampling evaluations across a wide range of scales. Given the usefulness, simplicity, and efficiency of training models to fill-in-the-middle (FIM), we suggest that future autoregressive language models be trained with FIM by default. To this end, we run a series of ablations on key hyperparameters, such as the data transformation frequency, the structure of the transformation, and the method of selecting the infill span. We use these ablations to prescribe strong default settings and best practices to train FIM models. We have released our best infilling model trained with best practices in our API, and release our infilling benchmarks to aid future research.

*Equal contribution, order randomized. Correspondence to: mobav@openai.com, heewoo@openai.com.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 1.1 | Our contributions | 4 |
| 2 | Evaluation | 5 |
| 2.1 | Autoregressive evaluation | 5 |
| 2.2 | Infilling evaluation | 6 |
| 3 | FIM training and inference | 6 |
| 3.1 | SPM mode | 7 |
| 3.2 | Context-level FIM | 7 |
| 4 | Pretraining results | 8 |
| 4.1 | Evaluation of left-to-right capabilities in downstream benchmarks | 9 |
| 4.2 | FIM rate | 9 |
| 4.3 | SPM vs PSM vs joint SPM+PSM training | 10 |
| 4.4 | Context-level vs document-level FIM | 12 |
| 4.5 | Middle span selection | 12 |
| 5 | Finetuning results | 13 |
| 6 | Discussion | 14 |
| 7 | Related work | 16 |
| 8 | Conclusion | 17 |
| 8.1 | Recommended FIM hyperparameters | 17 |
| 8.2 | Future directions | 17 |
| A | Architecture and datasets | 23 |
| B | Scaling trends for FIM rate ablations | 23 |
| C | Details of FIM implementation | 24 |
| D | Details of SPM encoding | 25 |
| E | Random span infilling benchmark | 25 |
| F | Dynamics and learning curves of finetuning | 26 |
| G | Top models comparison | 27 |
| H | Qualitative evaluation | 28 |
| H.1 | Successful infilling examples | 28 |
| H.2 | Limitations | 28 |
| H.3 | Mitigations | 29 |

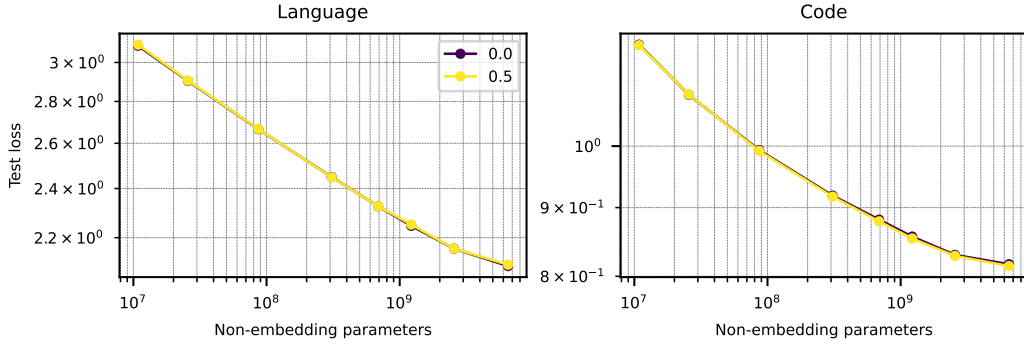


Figure 1: FIM can be learned for free. We pretrain language models with 50% and 0% FIM rates on two domains, natural language and code, and evaluate the test loss of all the final snapshots. All models are trained on 100B tokens of data. We observe that joint FIM training incurs no cost as the original left-to-right loss trend remains the same even though FIM models see the original data only 50% of the time and the models are learning a new capability. See Figure 3 for more evidence for the FIM-for-free property.

1 Introduction

Following the introduction of the Transformer [Vaswani et al., 2017], large language models (LLMs) trained on diverse Internet scale datasets have achieved remarkable success. These models are capable of producing coherent and sensible completions given a natural language prompt, and they achieve state-of-the-art performance in many benchmarks including reading comprehension, question answering, logical inference, and common sense reasoning.

There are several broad classes of transformer based language models: encoder-only models like BERT [Devlin et al., 2019] are typically trained with a masked language modeling objective, and encoder-decoder models like T5 [Raffel et al., 2019] are typically trained with a span prediction objective [Song et al., 2019]. Finally, causal decoder-based language models, like the GPT model series [Radford et al., 2018, 2019, Brown et al., 2020], are trained using the left-to-right next token prediction objective. The largest and most capable generative language models today, such as GPT-3, Codex, LaMDA, GLaM, PaLM, Gopher, Jurassic-1, and Chinchilla [Brown et al., 2020, Chen et al., 2021, Thoppilan et al., 2022, Du et al., 2021, Chowdhery et al., 2022, Rae et al., 2021, Lieber et al., 2021, Hoffmann et al., 2022], belong to the latter class of models. The overwhelming popularity of the causal decoder-based models at the largest scale is due to their superiority in open-ended text generation, in-context learning (using few-shot priming), pretraining computational efficiency [Wang et al., 2022], and to some extent historical precedence in successful scale-ups [Brown et al., 2020]. These models are also architecturally simpler and generally more effective without task specific finetuning, making them more attractive for inference and deployment.

All model classes are limited when it comes to infilling, where the model is tasked with generating text at a specific location within a prompt, while conditioning on both a prefix and a suffix. Left-to-right models can only condition on the prefix. While encoder-only and encoder-decoder models are capable of conditioning on suffixes, the lengths of infill regions seen at training time are typically much shorter than what is useful in practice. This is unfortunate because infilling naturally arises in applications where there is context both before and after the point of generation. For example, in creating a coding assistant, infilling can be used for docstring generation, import statement generation, or for completing a partially written function.

Our goal in this work is to address this limitation by adding fill-in-the-middle (FIM) capability to causal decoder-based language models which are currently the most dominant paradigm for large scale language modelling [Brown et al., 2020, Hoffmann et al., 2022, Chowdhery et al., 2022]. We show that with a simple modification to training data and without changing the model architecture, causal decoder-based autoregressive (AR) language models can learn infilling *without compromising their normal left-to-right generative capability*.

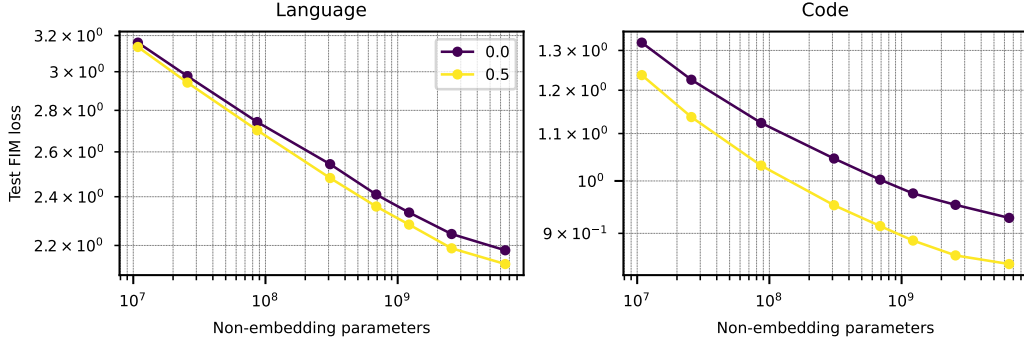


Figure 2: Evaluation of infilling capabilities of the same model scans from Figure 1 using FIM test losses. Models trained with FIM (yellow) obtain lower FIM test loss than baseline (purple) AR models. This shows that the FIM models are indeed learning to condition on the suffix while predicting the middle section allowing them to achieve lower test loss on FIM test set. Figures 1 and 2 together indicate that FIM models can be considered strictly better than AR models as they achieve the same left-to-right autoregressive loss but lower FIM loss.

The key to our approach, described in Section 3, is a transformation applied to a fraction of our dataset, in which we split documents into three pieces at random and move the middle piece to the end:

$$\text{document} \rightarrow (\text{prefix}, \text{middle}, \text{suffix}) \rightarrow (\text{prefix}, \text{suffix}, \text{middle})$$

We then concatenate the three pieces using sentinel tokens. This is similar to the procedure used in [Donahue et al., 2020, Aghajanyan et al., 2022, Fried et al., 2022].

Compared to prior work, our work emphasizes the computational efficiency of training FIM models. This emphasis is important given the increased interest in training very large language models, which are very expensive to train and have a substantial energy footprint. In general, when adding a new objective or capability to language models, we believe the most critical question is the effect on the existing capabilities and the computational efficiency trade-offs.

Unlike most cases where we jointly train on multiple objectives and datasets, we show that models trained jointly on a mixture of FIM transformed data and ordinary left-to-right data achieve the same left-to-right capability while learning how to fill-in-the-middle. We call this the FIM-for-free property.

In what follows, we use the term FIM model to refer to any model trained on a mixture of FIM transformed and normal left-to-right data. We refer to models trained without any FIM data (i.e. 0% FIM rate) as AR models.

1.1 Our contributions

Our central contributions in this paper are as follows:

- **FIM-for-free property:** We perform an extensive scaling study by training a suite of 8 models, with and without FIM, and show that FIM can be learned without compromising the left-to-right capability in pretraining. We examine this claim in both code and language, using both perplexity and sampling-based benchmarks.
- **Best practices for FIM in pretraining:** We clarify the effects of many hyperparameters related to training FIM models using comprehensive ablations. In particular, we study the FIM rate (the probability at which FIM transformation is applied to the data), different variants of FIM transformation, and the choice of middle span.
- **Finetuning inefficiency:** An alternative to training FIM models from scratch is to learn this capability by finetuning existing language models. We show that finetuning with FIM is computationally inefficient. While FIM can be learned for free during pretraining, learning FIM during finetuning requires a significant amount of additional compute to reach similar levels of performance as pretraining.

- **New infilling benchmarks.** In order to study the generative capabilities of our models, we need to evaluate the correctness of free-form generated samples. For this, we focus on code where we can use unit tests to evaluate the correctness of long FIM samples. In particular, we use the single-line and multi-line infilling benchmarks introduced by [Fried et al., 2022] by removing non-empty lines of canonical solutions of HumanEval [Chen et al., 2021]. However since line-based evaluations do not capture all the use cases of FIM, we create two new benchmarks called *random span infilling* and *random span infilling light*. We discuss these benchmarks and our evaluation methodology more generally in Section 2.
- **Need for sampling evaluations.** In Sections 4.2, 4.4, and Appendix B, we find that changing various hyperparameters in FIM training often leads to negligible differences in FIM test losses but large differences in sampling based benchmarks. Not only are sampling benchmarks closer to real use cases, but they also appear to be able to tease apart gains that can be missed using test losses. This is an important finding since often scaling laws analysis relies just on test losses, which we find are misleading if not augmented with other evaluations.

It is interesting to contrast the first and third bullet points above. The first states that learning FIM in pretraining is free while leaving it to finetuning is surprisingly costly. We discuss potential explanations for this finding in Section 6. To establish the FIM-for-free property, we perform an ablation study on both code and language across a range of scales. We train 8 models from 50M to 6.9B parameters, both with and without FIM, and compare the performance across a variety of autoregressive benchmarks. In particular, we train 16 models on code for 100B tokens and another 16 models on natural language for 100B tokens. The comparison of these models in terms of normal autoregressive left-to-right language modeling test loss is presented in Figure 1. In both domains, FIM models achieve similar AR test loss as the non-FIM models.

We provide more evidence for the FIM-for-free property by comparing FIM and AR models on non-loss based benchmarks in Section 4. Moreover, we see in Section 4.2 that there is a *stronger form of the FIM-for-free property*. Not only there is no hit in autoregressive capabilities from FIM training on the final checkpoints, the same also holds throughout training. This is evidenced by the matching learning curves between AR and FIM models in left-to-right loss and HumanEval evaluations in Figures 4 and 5.

Beside studying the effect of FIM training on the left-to-right capability, it is also important to show that the models are in fact learning to infill from FIM training. Figure 2 provides evidence for this in the context of FIM test losses. We study the infilling capabilities of our models more extensively in Section 4 and Appendix H.

2 Evaluation

We use both AR and FIM evaluation benchmarks to analyze the capabilities of our models. Vanilla AR evaluation is important for quantifying the impact of FIM training on left-to-right capabilities and allows us to demonstrate the FIM-for-free property from Section 1.1. FIM evaluation is important for understanding the effect of different hyperparameters on FIM training and to understand the scaling trends.

Throughout the paper, we use the terms AR and left-to-right interchangeably. AR loss refers to the cross entropy loss on normal left-to-right data and FIM loss as the loss on 100% FIM transformed data. All test losses are in nats per token unit. In all sampling-based benchmarks, we use nucleus sampling [Holtzman et al., 2020] with a nucleus parameter of 0.95.

2.1 Autoregressive evaluation

For all domains, we evaluate test losses in the canonical autoregressive order to show that the learning curves and scaling trends remain the same even with FIM augmentation. Beside test losses, we evaluate on standard benchmarks to demonstrate that the model’s capabilities are unaffected by FIM training. For natural language, we use PIQA [Bisk et al., 2020], Winograd [Levesque et al., 2012], WinoGrande [Sakaguchi et al., 2021] for common sense reasoning, DROP [Dua et al., 2019] and QuAC [Choi et al., 2018] for reading comprehension, and HellaSwag [Zellers et al., 2019], LAMBADA [Paperno et al., 2016], StoryCloze [Mostafazadeh et al., 2016] for completion tasks. All

benchmarks with the exception of DROP and QuAC are evaluated with few-shot prompting. For code, we measure the pass rates on HumanEval [Chen et al., 2021].

2.2 Infilling evaluation

To create FIM tests, we apply the FIM transformation to the examples from the AR test sets with a FIM rate of 100%. Using the same underlying examples in FIM and AR test sets allows us to compare FIM and AR test losses. Additionally, we create a masked version of these test sets where we only measure the loss on the middle span tokens. The latter test sets are used to measure $P(\text{middle}|\text{prefix}, \text{suffix})$ for FIM models and $P(\text{middle}|\text{prefix})$ for AR models allowing us to investigate the amount of information FIM models gain by being able to condition on the suffix.

For generative infilling capabilities, we focus on code since we are interested in free-form generation in contrast to single or few token generations common in cloze-style natural language benchmarks. The advantage of working with code is that we can use test suites to evaluate the correctness of samples in our tasks even when evaluating long samples from open-ended generations.

All the sampling based infilling benchmarks we use are partial function completions tasks created by removing middle spans from the canonical solutions of HumanEval [Chen et al., 2021]. In particular, we use the single-line and multi-line infilling benchmarks proposed by [Fried et al., 2022] where different spans of non-empty lines in the canonical solutions of HumanEval are turned into a FIM task. In addition, we create a new benchmark called random span infilling², where for each HumanEval problem, we create infilling tasks by selecting the middle span from the canonical solution uniformly at random. We show an example of such a task below where the model must predict the highlighted section (or an alternative completion accomplishing the same goal). We refer to Appendix E for more details.

```
def unique(l: list):
    """Return sorted unique elements in a list
    >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [0, 2, 3, 5, 9, 123]
    """
    return sorted(list(set(l)))
```

The single-line, multi-line, and random span infilling together constitute our infilling benchmark suite. These benchmarks have 1033, 5815, and 1640 tasks, respectively. We note that this is much larger than the number of tasks in the original HumanEval dataset (164 tasks), which reduces variance in our evaluations. Still, we take at least 100 to 200 samples per task to further reduce variance when evaluating these benchmarks on the final snapshots of our models. We also use random span infilling *light*, a smaller version of random span infilling, with only one random FIM task per HumanEval problem and just 164 tasks, to track the infilling capability trends during training.

In Section 3, we find that FIM can be prepared in two different ways denoted as PSM and SPM. We report just the SPM infilling results for brevity, except in cases when the use of PSM changes the conclusions.

3 FIM training and inference

We implement FIM using a random transformation applied to our dataset. We experiment with two different implementations: document level and context level. The difference between the two is at which stage of the data loading pipeline the FIM transformation occurs. This choice naturally arises because a long document can be broken into many contexts, or a context can contain multiple documents when the documents are small. We first describe the document-level case and then describe the changes required to implement context-level FIM in Section 3.2.

In document-level FIM, with a certain probability p called the **FIM rate** (we use $p = 0.5$ for our main suite of models), we cut each document into three parts: prefix, middle, and suffix. We perform this split prior to tokenization, when the document is still a sequence of characters. We split uniformly at random, which means the lengths of prefix, middle, and suffix are each 1/3 of the full document in expectation.

²Released at <https://www.github.com/openai/human-eval-infilling>

We then encode each of the three sections separately and prepend sentinel tokens to the beginning of each section. We denote these sentinel tokens by $\langle \text{PRE} \rangle$, $\langle \text{MID} \rangle$, and $\langle \text{SUF} \rangle$. Finally we concatenate all these sections in the order prefix, suffix, and middle along with their sentinel tokens to form the tokenized version of the FIM document,

$$\langle \text{PRE} \rangle \circ \text{Enc}(\text{prefix}) \circ \langle \text{SUF} \rangle \circ \text{Enc}(\text{suffix}) \circ \langle \text{MID} \rangle \circ \text{Enc}(\text{middle}), \quad (\text{PSM})$$

where \circ denotes concatenation. The different documents, whether FIM or AR, then are concatenated with $\langle \text{EOT} \rangle$ and are given to the model during training. We reiterate that we keep the loss on all three sections prefix, middle, and suffix, so FIM training does not cause a decrease in the autoregressive learning signal. Preliminary experiments, although not reported here, suggest that this choice is crucial for the FIM-for-free property to hold. This property does not change whether the sentinels are masked or not; however, it is important to always train on the $\langle \text{EOT} \rangle$ tokens as it signals a successful join to the suffix.

For inference, we encode the given prefix and suffix and prompt the model with

$$\langle \text{PRE} \rangle \circ \text{Enc}(\text{prefix}) \circ \langle \text{SUF} \rangle \circ \text{Enc}(\text{suffix}) \circ \langle \text{MID} \rangle. \quad (\text{PSM inference})$$

We continue sampling from the model until it generates the $\langle \text{EOT} \rangle$ token which is how the model communicates it has connected the prefix and the suffix.

If the model fails to generate an $\langle \text{EOT} \rangle$ token within a reasonable allotted inference token budget, it is often a sign the model is having a difficult time connecting the prefix and the suffix, and the resulting samples often will be of worse quality, which motivates the procedure of EOT aware best-of-n sampling. See Appendix H for more discussion.

3.1 SPM mode

We also introduce a variant of the above procedure where we swap the order of prefix and suffix, called SPM, to emphasize the changing of the order to suffix, prefix, and middle. Our main motivation for introducing SPM is improved key-value caching during inference. The reason for this advantage is that with SPM, appending tokens to the prefix no longer invalidates the keys and values computed in the suffix section. Note that superiority of SPM caching is not universal and may depend on the applications. In particular, in the SPM mode, minor changes to the suffix will invalidate the cache for prefix, but we expect changes to the suffix to be rarer than changes in prefix in real workloads. Interestingly, we find in Section 4.3 beside the caching advantages, SPM in fact has also a slight edge over PSM in the infilling benchmarks.

In our main runs, we apply the FIM transformation with 50% probability in PSM mode and with 50% probability in SPM mode, so the model is able to handle both types of formatting in inference. In other words, each mode inherits half of the total FIM rate p . We ablate this choice of joint training on PSM and SPM and compare with pure PSM and SPM runs. The results in Table 1 show the efficacy of this choice.

Even though the idea of SPM mode is simple, there are some subtleties with the placement of sentinel tokens in SPM which are especially important when training jointly on SPM and PSM. We describe these subtleties in Appendix D.

3.2 Context-level FIM

In language model training, documents are often joined with a boundary token, referred to as $\langle \text{EOT} \rangle$, and are then chunked to the model context length. When applying FIM to long documents, this operation can result in fragmented FIM data where the entire prefix or suffix could get cut out of the context during chunking. To address this issue, we can apply FIM after the chunking step. A context slice may have multiple documents in them joined with the $\langle \text{EOT} \rangle$ boundary token. So, we split based on $\langle \text{EOT} \rangle$, turn some of the documents into FIM examples with probability given by the FIM rate, and join the examples again with $\langle \text{EOT} \rangle$. The resulting slice is then trimmed to the model context length. We refer to Appendix C for more details of FIM transformation. In Section 4.4, we

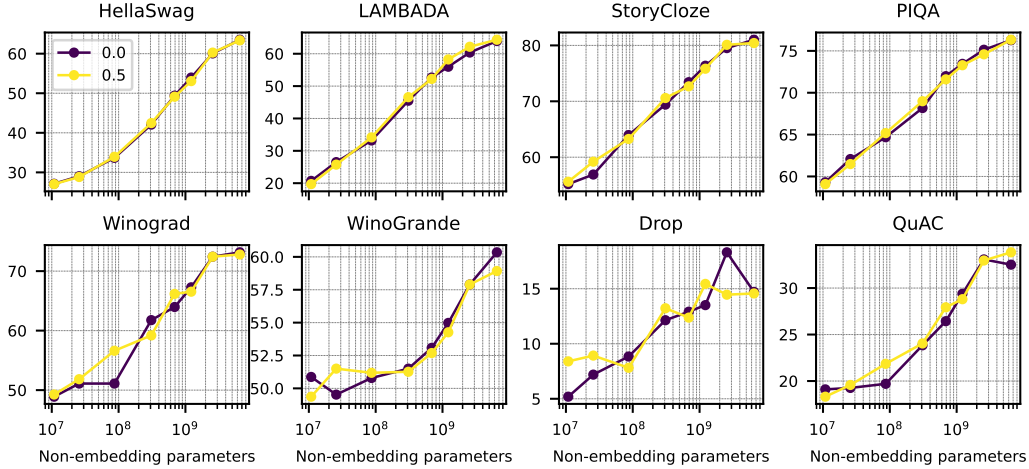
³It is worth noting that prepending this prompt with $\langle \text{EOT} \rangle$ leads to a slight improved performance, and we do so when evaluating our models in sampling benchmarks.

show this technique can boost performance relative to document-level FIM, and adopt context-level FIM in all our main FIM runs in this work.

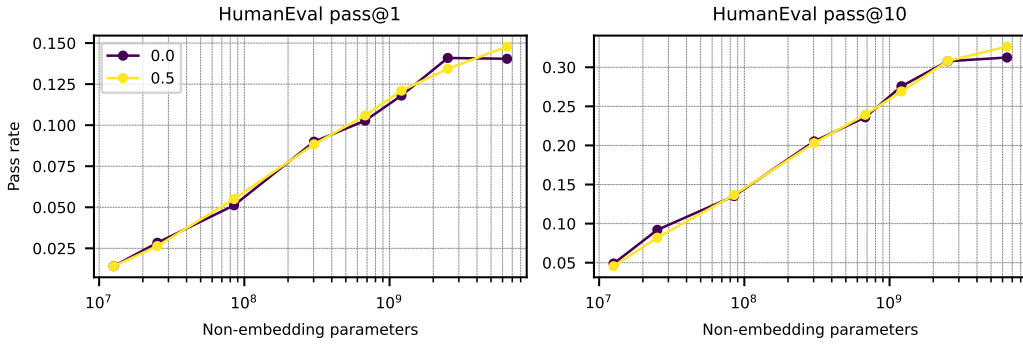
4 Pretraining results

In Section 1.1, we discussed the FIM-for-free property which states that FIM can be learned without any impact to the left-to-right capability. We start this section by presenting more evidence for this result. Next, we study the hyperparameters of FIM training including the FIM rate, PSM vs SPM vs joint training, context vs document-level FIM, and the choice of middle span. Although FIM is free from the point of view of AR capability, the FIM capabilities themselves depend strongly on these hyperparameters. We study these choices in the code domain, where we can measure the correctness of generated samples using unit tests.

The models, unless otherwise stated, are trained with fixed horizon of 100B tokens. For our main scans we use all the 8 models described in Appendix A. For more extensive scans, e.g. Sections 4.2, 4.4, and Appendix B, we use a subset of the models trained with a shorter horizon to limit the compute costs.



(a) Comparison of natural language results. We report F1 for Drop and QuAC and accuracy for the rest.



(b) Comparison of code results. We use temperature 0.8 and 400 samples per task for both pass@k.

Figure 3: Comparison of performance on standard benchmarks for the natural language (top) and code (bottom) domains. Joint training of next-token prediction and FIM allows the model to learn the new infilling task without affecting the original capabilities. This provides further evidence for FIM-for-free property.

4.1 Evaluation of left-to-right capabilities in downstream benchmarks

We train a series of models from 50M to 6.9B parameters from scratch with and without 50% FIM augmentation on natural language and code domains. Figure 1 shows that the left-to-right test loss is unaffected even though FIM models see the data in its original form half the time, and are simultaneously learning a new skill.

However, as we demonstrate below (see Sections 4.2 and 4.4) it is often not sufficient to just consider test loss. So to strengthen the above results, we evaluate our models on a suite of standard downstream benchmarks, the result of which is presented in Figure 3. We again find that joint FIM pretraining does not result in any degradation in standard AR benchmarks as the performance matches within error for both natural language and code.

4.2 FIM rate

From Figures 1 and 3, we see that a FIM rate of 50% incurs no performance hit in the left-to-right capabilities. This naturally raises several questions:

- Does FIM-for-free still hold even at higher FIM rates? If so, how high can we increase the FIM rate without degrading the left-to-right capabilities?
- Does using a higher FIM rate lead to stronger FIM capabilities? Or does the benefit saturate after a threshold?

In this section, we ablate the FIM rate to answer these questions. We train 6 large models (see Table 3) with FIM rates (0, 0.25, 0.5, 0.75, 0.9, 1.0) for 50B tokens. The results are presented in Figures 4 and 5. The left plot in Figure 4 provides evidence that a FIM rate even up to 90% does not cause any degradation in left-to-right capabilities. However, there is a clear sign of degradation in ordinary AR test loss with 100% FIM rate. For HumanEval, the left plot in Figure 5 shows all models irrespective of FIM rate have a similar performance.

On the other hand, we find that the FIM rate does significantly affect infilling capabilities. Even though the gain in FIM perplexity in Figure 4 due to a higher FIM rate is negligible, increasing this rate yields a consistent improvement in the infilling pass rate as shown in the right plot in Figure 5. This indicates that to investigate the FIM capabilities of our models, it is not sufficient to consider language modelling perplexity measures such as test loss, but we should also consider non-loss based evaluations.

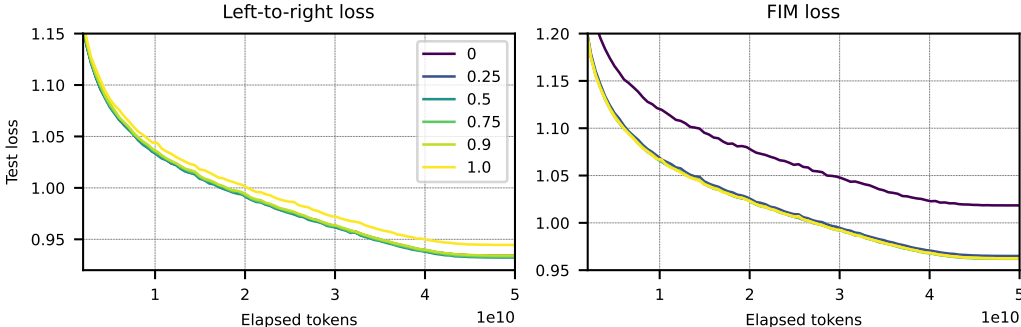


Figure 4: Comparison of the learning curves of large (see Table 3) models trained with different FIM rates for 50B tokens. A FIM rate even up to 90% does not have a noticeable effect on left-to-right test loss; however, at a FIM rate of 100% there is degradation. We can also see the *stronger FIM property* in the left figure: all runs with FIM rates less than 100% follow very closely to the original left-to-right test loss.

In Appendix B, we show further evidence across a range of scales that higher FIM rates improve infilling performance but that this gain is not reflected in the perplexity evaluation.

Given the results here and in Appendix B, it is natural to question why we train our core series of models with a FIM rate of 50% rather than 90% or higher. Models with a FIM rate of 90% show

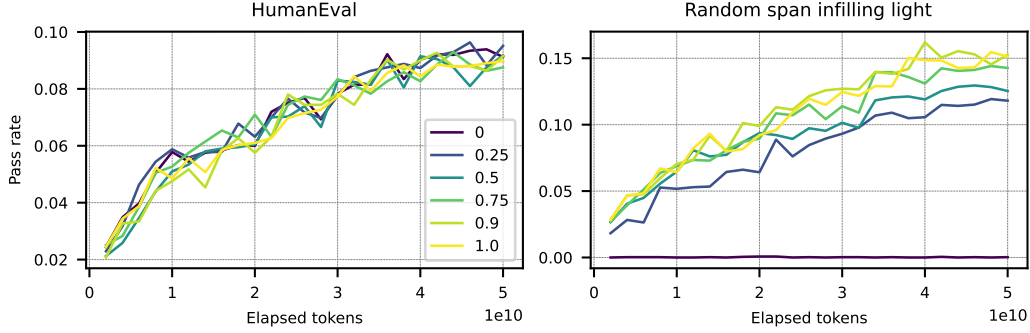


Figure 5: In-run evaluation of coding benchmarks with temperature 0.8 and 25 samples per task. Using higher FIM rates do not have a noticeable effect on HumanEval performance. A higher FIM rate shows stronger infilling capabilities on the light random span infilling benchmark.

superior performance while maintaining the FIM-for-free property. This was mainly accidental, as we had already trained the main series prior to seeing the FIM rate ablation results,⁴ and it was prohibitively costly to retrain all the models with the higher rate.

The results here motivated us to train a second 6.9B FIM model with a FIM rate of 90% on code to obtain the strongest infilling model to date at this scale. The comparison of results is found in Table 4. We note however from Figure 13 that a FIM rate of 50% does not seem to be too far from optimal.

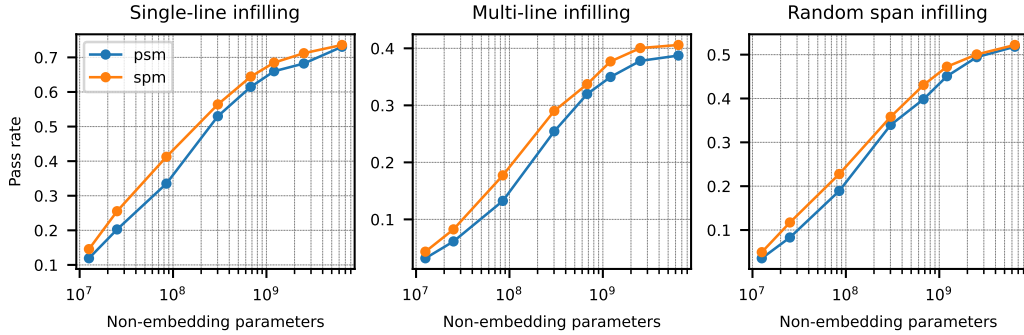


Figure 6: SPM mode shows a slight advantage in performance across scale. All the evaluations in this plot are at temperature 0.2 and 100 samples per task for single-line and multi-line infilling and 200 samples per task for random span infilling.

4.3 SPM vs PSM vs joint SPM+PSM training

In Section 3, we describe two ways of constructing a FIM example: [suffix, prefix, middle] and [prefix, suffix, middle]. Here we study how this choice affects performance during pretraining and evaluation.

The main finding is that SPM is slightly stronger than PSM in our benchmarks in general as evidenced by Figure 6. We train a series of FIM models with a FIM rate of 50% with the FIM rate equally allocated to PSM and SPM. We find that evaluating these models in SPM mode yields a consistently higher performance than PSM across scale. This is likely due to the fact that in SPM, there is no distinction between the prefix and the middle sections as they are one contiguous sequence of text. This makes it more natural for the model to continue from the prefix in contrast to PSM where attention has to first identify where the span token is.

⁴In particular, our earlier ablations based only on loss had indicated that the gains from increasing the FIM rate to 90% should be negligible, resulting in us choosing a more moderate value of 50%. More detailed study using all 3 infilling benchmarks showed that there is in fact a noticeable gain in using even a higher FIM rate.

However, this does not imply that we should train solely on SPM. In Table 1, we train large models on pure PSM, pure SPM, and our default 50-50 SPM+PSM mix, and evaluate them in all modes. We observe a positive transfer of capability between PSM and SPM. Training joint FIM with a 50% FIM rate obtains roughly the same performance in SPM mode as training pure SPM FIM with a 90% FIM rate. Not only is joint pretraining the most efficient, but it also yields the most flexible model with two inference modes.

It is noteworthy that the recent infilling works using data transformations similar to FIM such as [Donahue et al., 2020, Aghajanyan et al., 2022, Fried et al., 2022] utilize a format similar to PSM. The above findings indicate that this choice leads to suboptimal infilling performance.

| Train distribution | FIM rate | Single-line | | Multi-line | | Random span | |
|--------------------|----------|-------------|-------|------------|-------|-------------|-------|
| | | PSM | SPM | PSM | SPM | PSM | SPM |
| Joint | 0.5 | 0.550 | 0.595 | 0.265 | 0.293 | 0.367 | 0.379 |
| Joint | 0.9 | 0.616 | 0.622 | 0.290 | 0.305 | 0.397 | 0.420 |
| PSM | 0.9 | 0.583 | 0.625 | 0.273 | 0.305 | 0.362 | 0.274 |
| SPM | 0.9 | 0.023 | 0.586 | 0.008 | 0.301 | 0.007 | 0.386 |

Table 1: Comparison of FIM performance when trained and evaluated in various SPM, SPM settings. All the joint runs put 50% of the total FIM rate on PSM and 50% on SPM. All results are obtained with temperature 0.2 and 100 samples per task.

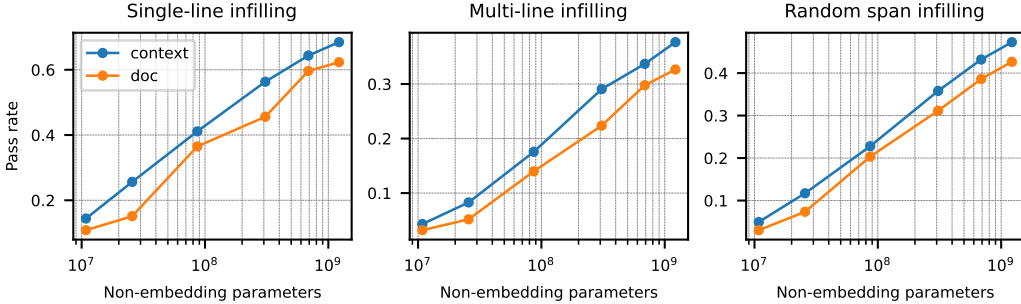


Figure 7: Applying FIM at the context level consistently outperforms document level FIM. All benchmarks are evaluated with temperature 0.2 and 200 samples/task.

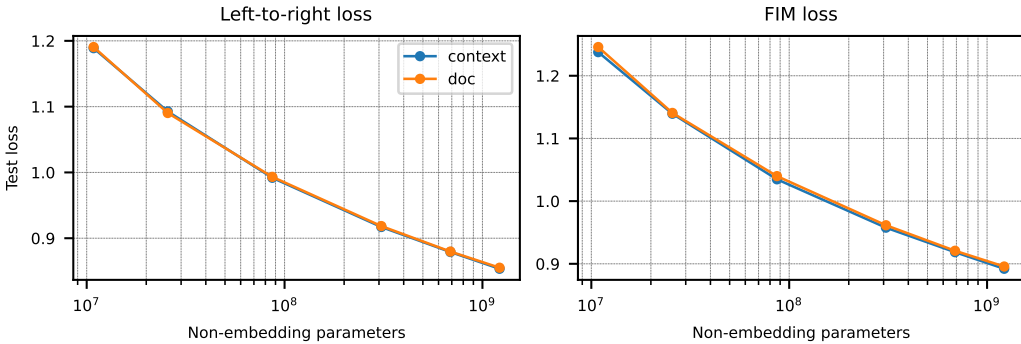


Figure 8: Comparison of losses with different FIM implementations. While document level FIM introduces partially broken data into training, it does not hurt the autoregressive loss (left). We also find that the reduction in FIM perplexity (right) is not commensurate to the gain in pass rate shown in Figure 7.

4.4 Context-level vs document-level FIM

In Section 3, we noted two ways of implementing FIM, context-level and document-level FIM, where augmentation is applied either before or after packing and chunking. We now ablate this choice on a series of code models trained with a 50% FIM rate and the default joint PSM-SPM mix.

In Figure 7, we find that context-level FIM yields a consistent and significant improvement over document-level FIM across all the range of scale. This is a noteworthy contrast to the perplexity evaluation in Figure 8 (right) where the improvement is an almost negligible 0.001 nats/token. This corroborates the finding in Section 4.2 that perplexity evaluation does not always capture the gains in the sampling performance.

Also, we previously explained that document-level FIM can result in fragmented FIM data with a missing prefix and/or suffix from the chunking step of data loading pipeline. Figure 8 (left) shows that training on these invalid examples in document-level FIM does not affect the left-to-right evaluation. Hence, practitioners might still sometimes prefer document-level FIM due to its simpler implementation.

4.5 Middle span selection

An important consideration in FIM training is the choice of middle span. In this work, the middle span is chosen uniformly at random where the split between prefix, middle, suffix happens at the character level. In this section, we examine this choice. Instead of trying FIM across syntactic boundaries, such as functions and class bodies, we restrict our ablations to simple, generalizable approaches which are language agnostic. We select spans in three different ways, splitting randomly by lines, tokens, and characters. The section boundaries are selected uniformly at random from the allowed splitting positions based on the span type. Here, a token refers to a word in the byte-pair encoding (BPE) vocabulary. In practice, this is implemented by applying the FIM augmentation after the documents are encoded with BPE (see Appendix C). For simplicity, we run all our experiments in PSM mode in this ablation.

In Table 2 we see that training only on the line-based middle spans gives the models a slight advantage in the single-line and multi-line infilling benchmarks. This is not surprising since these evaluations are completely in distribution with line based middle span runs. On the other hand, the line based training fails almost completely in the random span infilling benchmark. Interestingly, the advantage provided in line-based evaluations from concentrating all the FIM distribution on line based middle spans in training is quite small relative to how much it hurts the model in random span infilling benchmark.

Training with token-level random spans does slightly better on random span infilling, but is still not competitive compared to character-level runs on this benchmark. The reason is that token-level FIM models are never trained on cases where a token is broken into two parts across the boundaries of middle with prefix or suffix. When the middle section is selected completely at random at the character level, subtokens are introduced naturally at the beginning and the end boundaries of the middle section. There is no train-test mismatch and the model is able to understand and solve more random span infilling tasks while still performing well in single-line and multi-line infilling.

| Training middle span | Single-line infilling | Multi-line infilling | Random span infilling |
|-----------------------------|-----------------------|----------------------|-----------------------|
| Line-level random span | 0.586 | 0.269 | 0.015 |
| Token-level random span | 0.548 | 0.242 | 0.102 |
| Character-level random span | 0.557 | 0.250 | 0.321 |

Table 2: Pass rates of medium models pretrained with various middle span selection strategies. Training on line-based spans improves the single- and multi-line infilling metrics reported in InCoder, but line- and token-level spans used in previous works can not robustly handle real life use cases where the span starts or ends in subtokens. Overall, character-level random span run dominates in random span benchmark while it is also not far behind in single and multi line infilling.

5 Finetuning results

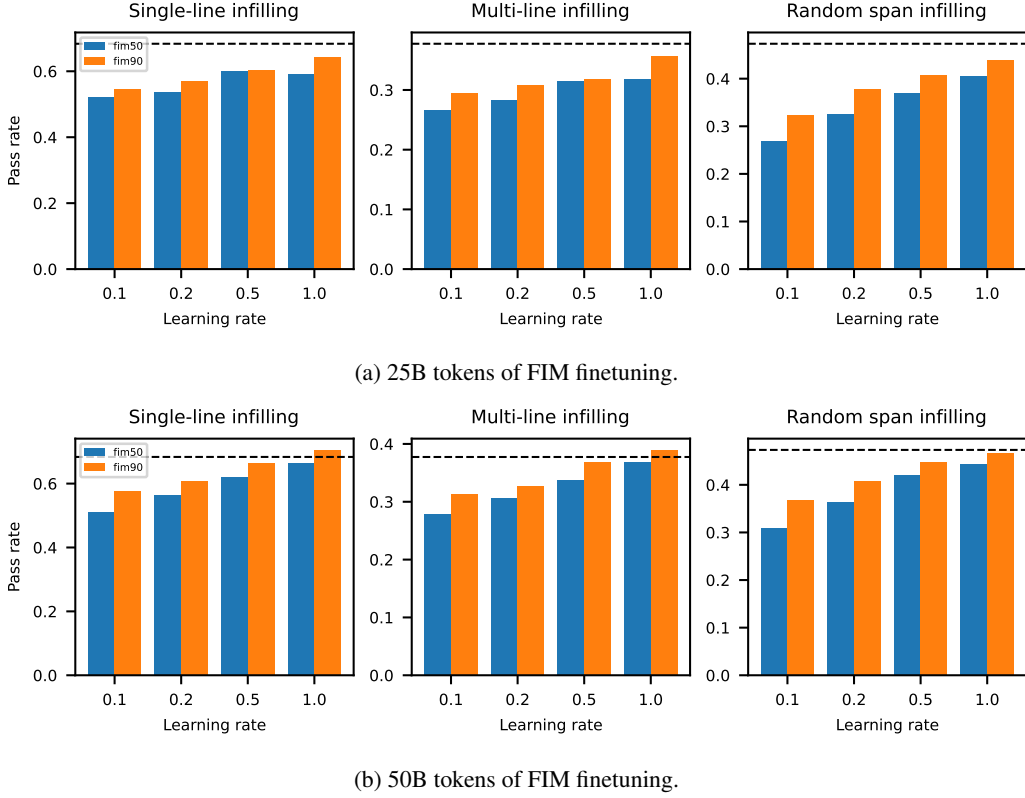


Figure 9: Evaluation of the final snapshots of models pretrained for 100B tokens without FIM and then finetuned for 25B (row a) and 50B (row b) tokens with FIM. The x-axis shows the learning rate multiplier relative to the pretraining learning rate. The dashed line indicates the baseline performance of the model pretrained for 100B tokens with a FIM rate of 50% with no additional finetuning. Only the most aggressive combination of 90% FIM rate and a learning rate multiplier of 1.0 with 50B tokens of finetuning catches up to the performance of the baseline. Reported results are with temperature 0.2 and 100 sampler per task.

In this section, we investigate whether we can finetune existing AR models to learn the FIM capability. Ideally, after finetuning, an AR model would reach the same level of performance on FIM evaluations as it would have achieved if it were pretrained with FIM. Given that FIM can be learned during pretraining without extra compute cost, it is natural to expect that the model should also be able to learn this task quickly in finetuning. Surprisingly, we find that for finetuned models to reach the same level of performance as baseline pretrained models, one needs to expend a large amount of compute relative to the pretraining compute.

To show this, we finetune an XL model pretrained for 100B tokens without FIM using different choices of finetuning hyperparameters. Specifically, we train 16 finetuned models with 4 choices of learning rates (0.1, 0.2, 0.5, 1x multiples of pretraining learning rates), 2 different FIM rates (0.5 and 0.9), and 2 different choices of finetuning horizons (25B and 50B tokens). We use this large variety of hyperparameter choices to both ensure that our conclusion is robust and to better understand the effect of hyperparameters on the final performance. The results are summarized in Figure 9 where we compare the performance of these 16 models with that of the XL model trained for 100B tokens with a FIM rate of 50% without any finetuning. It is evident from this figure that even with significant additional finetuning compute, AR models finetuned with FIM do not reach the same performance as the models pretrained with FIM (and without any FIM finetuning).

Among these 16 models, the only setting where the gap between pretrained baseline and finetuned models is closed is the 50B token run with a FIM rate of 0.9 and learning rate multiplier of 1.0 relative

to pretraining. More generally, we find that higher learning rate, FIM rate, and longer finetuning all seem helpful for improving FIM performance in finetuning.

We find it particularly surprising that such high learning rates and lengthy finetuning are necessary for reaching the similar level performance. We discuss this topic more in Section 6. We note that although reaching the same level of performance as in pretraining requires a large amount of compute, a small amount of finetuning (especially with high FIM and learning rate) is still sufficient for the model to reach non-trivial levels of FIM performance on our metrics. We present further results on dynamics of finetuning in Appendix F.

6 Discussion

Pretraining vs finetuning. In the previous sections, we studied how to efficiently teach FIM to causal language models. A main finding was that FIM can be learned for free in pretraining. In contrast, we saw in Section 5 that learning FIM in finetuning can be quite expensive. Here we describe some potential explanations for these findings.

The main intuition for why FIM can be learned for free in pretraining is that breaking a document into three pieces and shifting the middle one to the end effectively creates three smaller documents. In particular, each piece still requires predicting next tokens from left to right, keeping the total number of tokens processed autoregressively the same.

On the other hand, even though FIM data is locally identical to autoregressive data, FIM does impose a different global attention pattern over the whole document. To visualize this, we show the causal attention mask of a FIM document in Figure 10. These new attention pattern could be the reason why it takes a relatively long token horizon and a high learning rate to learn FIM in finetuning. It is possible that there is ossification [Hernandez et al., 2021] in the learned document-wide attention pattern in regular AR pretraining which requires a lengthy finetuning stage to adapt to the attention pattern needed in FIM.

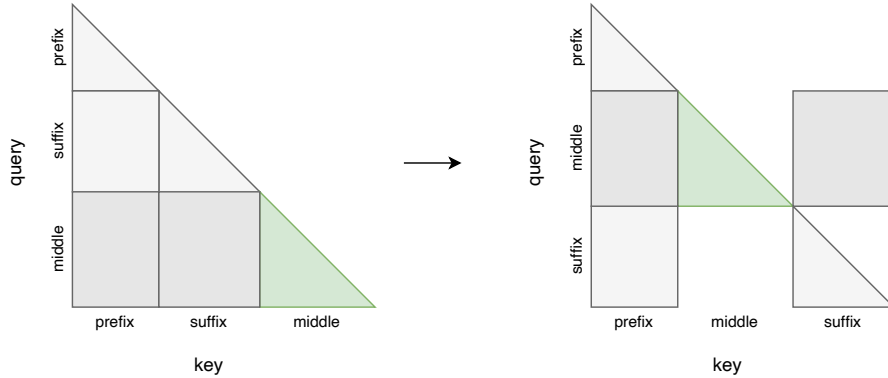


Figure 10: Visualization of causal attention pattern of FIM data. Unraveling both the query and key embeddings back in the canonical left-to-right order shows that FIM allows the transformer to attend to future context when decoding the middle section without complex architectural changes. One side-effect is that the suffix probability no longer depends on the middle span.

FIM loss, AR loss, and the difficulty of FIM task. Naively, since FIM does not come at a cost in AR capability, one may expect FIM to be an easy task. In fact, the opposite seems to be the case. There is substantial evidence that FIM can often be much harder than normal left-to-right generation.

Intuitively, it is often easier to continue a text in a plausible manner than to continue the text conditioned on ending in a specific suffix. The latter requires planning a plausible narrative connecting the two pieces, starting the generation in a way that matches the prefix, and *stopping the generation at the right time* so it connects to the suffix. In particular, in FIM the model is trained to generate `<EOT>` when the middle ends and connects to the suffix. On the other hand, when the model fails to produce `<EOT>` in the allotted budget, we often end up with truncated samples which do not connect well to the suffix. For example, consider the following:

When I was young, I only liked to play video games. Over time, I started thinking if it'd be possible to make bots to play better than any human can ever play these games. I eventually decided I liked working on the latter more than playing the games themselves and that's how first I got interested in AI research.

When I was young, I only liked to play video games. I would play sometimes more than 13 hours per day. The rush, novelty, and variety were beyond anything real life could offer. I loved the challenge and I excelled at it. I would often skip classes and go to and that's how first I got interested in AI research.

Both completions above connect well to the prefix, but only the first manages to connect well to the suffix. The second completion in contrast fails to produce `<EOT>` in the allotted budget resulting in a bad sample.⁵ This turns out to be a common failure in FIM sampling. Even though, left-to-right sampling also struggles sometimes with related issues, this type of failure is more troublesome in FIM since a failure to connect to the suffix cannot easily be fixed by post-processing. For example, trimming the sample to the last paragraph or line is often an effective way in improving sample quality in AR sampling, but does not help in FIM. We discuss this and other issues associated with FIM sampling more extensively in Appendix H.

The difficulty of FIM task compared to AR task is also reflected in the loss associated with each task. To see this, in Figure 11, we compare the FIM loss with the AR loss over a suite of FIM models all with 50% FIM rate. To remove confounders, we ensure the documents that underlie the AR test set are the same documents that are transformed through FIM to make up the FIM test set. We find the FIM perplexity is consistently higher than the AR perplexity across scale. That is, on average

$$P_{\text{FIM}}([\text{prefix}, \text{suffix}, \text{middle}]) < P_{\text{AR}}([\text{prefix}, \text{middle}, \text{suffix}]),$$

which means the models have a harder time modelling the same document in FIM format than AR format.

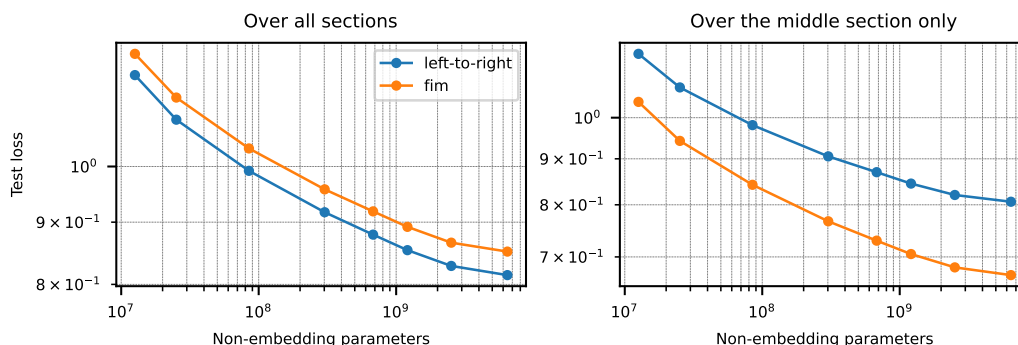


Figure 11: Comparison of the overall (left) and middle span (right) loss of 50% FIM code models. In the left plot, we see that the AR loss is consistently lower than the FIM loss suggesting that next-token prediction is inherently more compressible than infilling in the middle. The right figure evaluates the conditional loss of the middle span given the surrounding context showing that $P_{\text{FIM}}(\text{middle}|\text{prefix}, \text{suffix}) > P_{\text{AR}}(\text{middle}|\text{prefix})$. Here, FIM attains a lower loss because it can attend to the suffix. We emphasize that left-to-right and FIM here do not refer to model type, as all models in this figure are FIM models. They refer rather to the type of test loss used in evaluation.

Context-level vs document-level FIM and FIM rate. In Section 4.4, we saw that context-level FIM typically outperforms document-level FIM. Here, we note a connection between this finding and the results in Section 4.2 and Appendix B about FIM rate.

The basic observation is that document-level FIM effectively leads to a *lower FIM rate* compared to context-level FIM, even with the same nominal value of FIM rate. As a thought experiment, consider

⁵Even though the completion may have been able to connect to the suffix with a bigger budget, the challenge is it is unclear how much budget is enough. In practice, often a reasonable budget for the maximum number of tokens for the middle must be imposed.

the setting where all the documents in the training dataset are much longer than the context size. In this setting, when using document-level FIM, the model almost never sees the prefix, middle, and suffix of the same document appear in the same context together after chunking. As such, we would expect the model to struggle to learn FIM in this setting. In less extreme situations, there are many documents shorter than the context size and hence the above phenomenon is less pronounced. Still, because of long documents in training data and the usual artifacts of document packing, document-level FIM results in a lower effective FIM rate. Here, we define the effective FIM rate as the fraction of examples that are in FIM format and with all three of the prefix, middle, and suffix appearing within the same context.

This decrease in effective FIM rate is likely the main reason behind the stronger performance of context-level FIM in Section 4.4. We note that the exact amount of decrease in effective FIM rate depends on the details of distribution of document lengths. It is important to remember that even if the data distribution does not have many long examples, the decrease in effective FIM rate will still be present because of document packing.

7 Related work

Masked language modeling is closely related to text infilling in that consecutive runs of masked tokens can be interpreted as spans that the model must infill. While early masked language models like BERT [Devlin et al., 2019] masked tokens randomly, T5 [Raffel et al., 2019], SpanBERT [Joshi et al., 2020], and BART [Lewis et al., 2020] demonstrated improvements when contiguous runs of tokens are masked. However, because these models focus on representation learning, the span lengths are typically much shorter than a sentence or even a single line of code. Within our modalities of interest, DOBF [Lachaux et al., 2021] trains BERT on code, and HTLM [Aghajanyan et al., 2021] trains BART on HTML data.

Text infilling can also be seen as a special case of autoregressive language modeling where the standard left to right generation order is replaced by a more flexible ordering. XLNet [Yang et al., 2019] modifies the attention mask in a standard transformer to enable token generation in any user-specified order, while Insertion Transformer [Stern et al., 2019], KERMIT [Chan et al., 2019], and InDIGO [Gu et al., 2019] allow the model to predict a location for the next token before predicting the token. Similarly, Blank Language models [Shen et al., 2020] generate text by iteratively selecting a blank and replacing it with a token (and optionally more blanks).

Similar to our work, Zhu et al. [2019], Donahue et al. [2020], GLM [Du et al., 2022], CM3 [Aghajanyan et al., 2022], and InCoder [Fried et al., 2022] utilize left-to-right autoregressive modeling by moving the infill regions to the end of context, with regions separated by sentinels. Notably, Donahue et al. [2020] explore infilling spans of varying granularities, such as words, sentences, or paragraphs, and InCoder [Fried et al., 2022] uses a similar evaluation framework to ours by studying infilling capabilities on sampling based benchmarks created from HumanEval [Chen et al., 2021]. While several of these works support infilling multiple spans, we focus on the single span setting for practicality (e.g. in computer-based text generation, where the placement of cursor implies the location we want to infill). Additionally, our paper emphasizes the computational efficiency of training for infilling at scale. While we do not study syntactically or semantically motivated infilling spans, we show selecting spans at the character level improves the robustness of infilling.

Text infilling can also be performed using a GAN [Fedus et al., 2018], but REINFORCE is required to deal with the discreteness of text. Text infilling can also be done through gradient search [Liu et al., 2019], where tokens within the infilled span are optimized with gradient descent and collapsed to the nearest neighbor.

Overall, there are two approaches for imbuing models with infilling capabilities: first, through new architectures like SpanBERT and XLNet; second, through data formatting. In general, the latter approach can be seen as altering the behavior of a language model through control codes, which was motivated in CTRL [Keskar et al., 2019] to improve the steerability of generation. DistAug [Jun et al., 2020] is another related work that trains jointly on transformed data while conditioning on the transformation type. While infilling is a specific use case that can be realized through both architecture and data, it is generally easier and more universal to learn additional skills by introducing new training distributions than hardwiring them.

The strongest infilling system at scale to our knowledge currently is code-davinci-002 released this past March [OpenAI et al., 2022]. The present paper describes some of the early research that went into powering the infilling capabilities of this more powerful model. In Appendix 4, we present a comparison between this system, our 6.9B models, and the InCoder 6.7B model on our infilling benchmarks.

8 Conclusion

In this work, we show that causal decoder-based language models can learn to fill in the middle of a document after being jointly trained on a mixture of traditional left-to-right and FIM transformed data. A single FIM model can import modules, write docstrings, and complete functions, subsuming specialized models finetuned for individual tasks [Chen et al., 2021], providing substantial extra capability over traditional left-to-right language models.

One important finding here is the FIM-for-free property. Figures 1 and 2 show that with the same amount of compute, FIM models achieve the same test loss as AR models on left-to-right test loss while achieving lower FIM loss. This is further strengthened using non-loss based evaluations in Section 4.

We also investigate FIM finetuning since a lot of the existing language models do not have FIM capabilities. Our results demonstrate that a canonically pretrained left-to-right model does not acquire the new skill to the fullest extent of the given model size even with careful hyperparameter tuning and a significant amount of finetuning compute relative to pretraining. This suggests that for the best FIM performance, pretraining jointly from scratch with our recommended hyperparameters is more effective than finetuning.

To study FIM capabilities precisely, we use the infilling code benchmarks from InCoder [Fried et al., 2022] and introduce the new random span infilling benchmarks based on HumanEval [Chen et al., 2021]. From these, we learn a few important lessons. First, perplexity does not reflect the true infilling performance, and one should design the infilling benchmarks carefully to measure progress. Second, FIM capabilities depend considerably on the FIM rate and implementation like context-level FIM but left-to-right capabilities are unaffected by these choices as long as the FIM rate is kept below 100%. Third, applying FIM at the character level imbues the model with natural robustness to subtokens and makes it possible to deploy the model in the wild, for example, as a coding assistant.

All in all, we show FIM models are strictly more capable than canonically trained left-to-right models, at least within the bounds of the evaluations we consider, and we demonstrate how to train FIM models efficiently and competitively.

8.1 Recommended FIM hyperparameters

In Section 4, we see there are a number of hyperparameters in training FIM models. In all cases, we recommend applying FIM transformation at the character level and always including some character-level random spans as it allows the model to generate sensible completion even when the prefix and suffix end in the middle of a token. We note that for mid-token robustness, inference in PSM mode can be superior to the particular SPM mode explored in this work. However, pretraining with joint PSM and SPM yields the best performance due to a positive transfer between the two formats. In terms of implementation, context-level FIM is superior but document-level FIM is also an option if a simpler implementation is desired. Finally, we observe improved performance even up to a FIM rate of 90% without any cost in AR capabilities. In practice, any value in the range between 50% and 90% is a reasonable choice. Note that this is in contrast with some related prior work such as [Fried et al., 2022] which typically uses lower values of FIM rate such as 15%, which our results indicate to be suboptimal.

8.2 Future directions

There are several important related directions that we did not cover here. For example,

1. **Smarter span selection:** We only consider spans selected uniformly at random for generality, but mixing in semantically or syntactically meaningful spans [Donahue et al., 2020, Joshi et al., 2020, Deng et al., 2021] can considerably improve infilling performance. In

Section 4.5, we see that training on line-level spans instead of character-level spans improves line-based infilling results. In our preliminary experiment, selecting the middle span to be exactly one word was shown to significantly improve accuracy on cloze-like tasks. Smarter span selection involves language specific parsing and new benchmarks which may be tricky to make, but we expect this to produce stronger FIM models.

2. **Steerable generation:** FIM models generate spurious content or struggle to generate a sensible completion in the allotted token budget because they do not know the length or the style of infilling the user desires. Applying ideas like RL from human feedback [Stiennon et al., 2020] and instruction following [Ouyang et al., 2022], among other methods of controllable generation, could address this issue by providing further alignment with the users’ intent.
3. **Further examination of the FIM-for-free property:** Even though we provide substantial evidence for the FIM-for-free property, we cannot completely rule out that there are benchmarks not considered here where FIM models underperform AR models. As such, further strengthening or refuting the FIM-for-free property remains an interesting direction.
4. **Multiple infilling slots:** Many prior works in infilling explored multiple infilling slots [Raffel et al., 2019, Fried et al., 2022]. We do not study this, as there are already a number of considerations in training single-slot models, and inference challenges unique to infilling. Furthermore, in most applications, we anticipate single-slot infilling to be just as useful. We anticipate the inference challenges and failure modes to increase when considering multi-slot infilling. To make progress in multi-slot infilling however, creating appropriate sampling-based benchmarks is essential, as perplexity based evaluation would be increasingly unhelpful. There is a vast design space for these benchmarks and a vast array of extra training hyperparameters when going from single-slot to multi-slot infilling.
5. **Improving natural language FIM performance:** Qualitatively, our FIM models tend to perform better in code than language. This is perhaps not surprising given that code is a formal language, and as such, has more structure and less uncertainty. Improving infilling performance on natural language is an interesting future direction, but can be tricky because evaluation of free-form generation in language is not as straightforward as measuring functional correctness in code. We expect training on more semantically meaningful or shorter spans can help here but it is unclear what test distribution to use and how to evaluate this well in the general case.
6. **Role of bidirectionality and attention:** There is much to be understood in the role of attention and the training objective in free-form infilling performance. In this work, we use decoder based language models, which are currently the dominant paradigm of large scale language modelling. However, it is possible that from the point of view of infilling, other training objectives and architectures are superior. In this direction, [Artetxe et al., 2022] show a BERT style architecture performs better than FIM-like models but the results are mostly limited to single-token infilling. A more systematic study, similar to [Wang et al., 2022, Tay et al., 2022] but focused on free-form infilling generation, can clarify this further. Somewhat related to this, it is interesting to investigate the interaction of absolute and relative positional embedding and their variants with FIM. Preliminary results, not reported here, indicate that the FIM-for-free property still holds with absolute positional embedding.

Finally, our experience with the FIM-for-free property brings up the intriguing question of *what other useful skills can be learned jointly with no or little cost to the original capabilities of language models*. There have been a number of interesting works on this topic and we anticipate even more to follow, but many works often omit critical analysis for more broad adoption and comparison. We propose the following methodology to help advance research toward answering this question:

1. Establish a *budget* in the amount of original capabilities that one is willing to sacrifice to learn a new capability.
2. Maximize the new capability within this budget.

The budget-capability trade-off is not only theoretically interesting but also practical, allowing researchers to integrate new capabilities based on proper trade-off analysis. We look forward to a future where large language models have increasingly diverse and high value capabilities.

Acknowledgments

We would like to thank Shantanu Jain, Alex Paino, Alec Radford, Nick Ryder, Pranav Shyam, and Qiming Yuan for useful discussions and help at various stages of the project. We are also grateful to Christina Kim, Rachel Lim, Andrew Mayne, Maddie Siemens, and Natalie Staudacher for the help with the API infrastructure and qualitative evaluation of FIM, and to Angela Jiang, Katie Mayer, Rajeev Nayak, Henrique Pondé, and Felipe Such for invaluable work and immense effort on deployment. Finally, we thank Bob McGrew and Wojciech Zaremba for unceasing support throughout the project, and Karl Cobbe, Angela Jiang, Alec Radford, and Pranav Shyam for their valuable feedback on the paper.

References

- A. Aghajanyan, D. Okhonko, M. Lewis, M. Joshi, H. Xu, G. Ghosh, and L. Zettlemoyer. HTLM: hyper-text pre-training and prompting of language models. *CoRR*, abs/2107.06955, 2021. URL <https://arxiv.org/abs/2107.06955>.
- A. Aghajanyan, B. Huang, C. Ross, V. Karpukhin, H. Xu, N. Goyal, D. Okhonko, M. Joshi, G. Ghosh, M. Lewis, and L. Zettlemoyer. CM3: A causal masked multimodal model of the internet. *CoRR*, abs/2201.07520, 2022. URL <https://arxiv.org/abs/2201.07520>.
- M. Artetxe, J. Du, N. Goyal, L. Zettlemoyer, and V. Stoyanov. On the role of bidirectionality in language model pre-training, 2022. URL <https://arxiv.org/abs/2205.11726>.
- Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.
- T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- W. Chan, N. Kitaev, K. Guu, M. Stern, and J. Uszkoreit. KERMIT: generative insertion-based modeling for sequences. *CoRR*, abs/1906.01604, 2019. URL <http://arxiv.org/abs/1906.01604>.
- M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL <https://arxiv.org/abs/2107.03374>.
- E. Choi, H. He, M. Iyyer, M. Yatskar, W.-t. Yih, Y. Choi, P. Liang, and L. Zettlemoyer. QuAC: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2174–2184, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1241. URL <https://aclanthology.org/D18-1241>.
- A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL <https://aclanthology.org/P19-1285>.

- X. Deng, Y. Su, A. Lees, Y. Wu, C. Yu, and H. Sun. Reasonbert: Pre-trained to reason with distant supervision. *CoRR*, abs/2109.04912, 2021. URL <https://arxiv.org/abs/2109.04912>.
- J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *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, 2019. doi: 10.18653/v1/n19-1423. URL <https://doi.org/10.18653/v1/n19-1423>.
- C. Donahue, M. Lee, and P. Liang. Enabling language models to fill in the blanks. *CoRR*, abs/2005.05339, 2020. URL <https://arxiv.org/abs/2005.05339>.
- N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. *arXiv preprint arXiv:2112.06905*, 2021.
- Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang. 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)*, pages 320–335, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.26. URL <https://aclanthology.org/2022.acl-long.26>.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL <https://aclanthology.org/N19-1246>.
- W. Fedus, I. Goodfellow, and A. M. Dai. Maskgan: Better text generation via filling in the_____, 2018. URL <https://arxiv.org/abs/1801.07736>.
- D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer, and M. Lewis. InCoder: A generative model for code infilling and synthesis, 2022. URL <https://arxiv.org/abs/2204.05999>.
- J. Gu, Q. Liu, and K. Cho. Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7:661–676, 2019. doi: 10.1162/tac1_a_00292. URL <https://aclanthology.org/Q19-1042>.
- D. Hernandez, J. Kaplan, T. Henighan, and S. McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.
- J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *ICLR*. OpenReview.net, 2020. URL <http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#HoltzmanBDFC20>.
- M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tac1_a_00300. URL <https://aclanthology.org/2020.tac1-1.5>.
- H. Jun, R. Child, M. Chen, J. Schulman, A. Ramesh, A. Radford, and I. Sutskever. Distribution augmentation for generative modeling. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5006–5019. PMLR, 13–18 Jul 2020. URL <https://proceedings.mlr.press/v119/jun20a.html>.

- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. CTRL: A conditional transformer language model for controllable generation. *CoRR*, abs/1909.05858, 2019. URL <http://arxiv.org/abs/1909.05858>.
- M. Lachaux, B. Rozière, M. Szafraniec, and G. Lample. DOBF: A deobfuscation pre-training objective for programming languages. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 14967–14979, 2021. URL <https://proceedings.neurips.cc/paper/2021/hash/7d6548bdc0082aacc950ed35e91fcccb-Abstract.html>.
- H. J. Levesque, E. Davis, and L. Morgenstern. The Winograd Schema Challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, KR’12*, pages 552–561. AAAI Press, Rome, Italy, 2012. ISBN 978-1-57735-560-1. URL <https://cs.nyu.edu/faculty/davise/papers/WSKR2012.pdf>.
- M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. 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*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL <https://aclanthology.org/2020.acl-main.703>.
- O. Lieber, O. Sharir, B. Lenz, and Y. Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 2021.
- D. Liu, J. Fu, P. Liu, and J. Lv. TIGS: An inference algorithm for text infilling with gradient search. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4146–4156, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1406. URL <https://aclanthology.org/P19-1406>.
- N. Mostafazadeh, N. Chambers, X. He, D. Parikh, D. Batra, L. Vanderwende, P. Kohli, and J. Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1098. URL <https://aclanthology.org/N16-1098>.
- OpenAI, M. Bavarian, A. Jiang, H. Jun, and H. Pondé. New GPT-3 Capabilities: Edit and Insert. *OpenAI blog*, 2022. URL <https://openai.com/blog/gpt-3-edit-insert/>.
- L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022. URL <https://arxiv.org/abs/2203.02155>.
- D. Paperno, G. Kruszewski, A. Lazaridou, N. Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL <https://aclanthology.org/P16-1144>.
- I. Provilkov, D. Emelianenko, and E. Voita. Bpe-dropout: Simple and effective subword regularization. *CoRR*, abs/1910.13267, 2019. URL <http://arxiv.org/abs/1910.13267>.
- A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.
- A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

- J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.
- C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.
- K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, aug 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL <https://doi.org/10.1145/3474381>.
- P. Shaw, J. Uszkoreit, and A. Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.
- T. Shen, V. Quach, R. Barzilay, and T. Jaakkola. Blank language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5186–5198, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.420. URL <https://aclanthology.org/2020.emnlp-main.420>.
- K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019.
- M. Stern, W. Chan, J. Kiros, and J. Uszkoreit. Insertion transformer: Flexible sequence generation via insertion operations. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5976–5985. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/stern19a.html>.
- N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize from human feedback. *CoRR*, abs/2009.01325, 2020. URL <https://arxiv.org/abs/2009.01325>.
- Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, D. Bahri, T. Schuster, H. S. Zheng, N. Houlsby, and D. Metzler. Unifying language learning paradigms, 2022. URL <https://arxiv.org/abs/2205.05131>.
- R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- T. Wang, A. Roberts, D. Hesslow, T. L. Scao, H. W. Chung, I. Beltagy, J. Launay, and C. Raffel. What language model architecture and pretraining objective work best for zero-shot generalization?, 2022.
- Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL <https://aclanthology.org/P19-1472>.
- W. Zhu, Z. Hu, and E. P. Xing. Text infilling. *CoRR*, abs/1901.00158, 2019. URL <http://arxiv.org/abs/1901.00158>.

A Architecture and datasets

We use 8 causal transformer decoder models [Vaswani et al., 2017] with similar architecture, optimization hyperparameters, and encoding to Codex and GPT-3 [Chen et al., 2021, Brown et al., 2020]. The main architectural details of our models are summarized in Table 3. The only architectural modification we introduce is the use of relative attention [Shaw et al., 2018, Dai et al., 2019] rather than learned positional embeddings. This increases the parameter count negligibly but leads to improved performance. We also increase the learning rates of our three largest models by a factor of 2 for improved final performance, as it is known that GPT-3 series of models use rather conservative choices of learning rates. The context size for all the models is 2048.

We train our code models on the same dataset that was used to train Codex, which is a 159 GB Python dataset scraped in May 2020. As such, we expect no train set contamination from the subsequent public release of HumanEval. Similar to GPT-3 and unlike Codex, we train our models from scratch from a random initialization. All models from the main scans are trained for 100B tokens irrespective of size. Due to this fixed token budget, we expect our largest models to be undertrained [Hoffmann et al., 2022] and to benefit significantly from longer training. For our natural language models, we use the same dataset as was used in GPT-3 [Brown et al., 2020], the details of which are described in Section 2.2 of that paper.

| Model Name | n_{param} | n_{ne} | n_{layers} | d_{model} | n_{heads} | d_{head} | Batch Size | Learning Rate |
|------------|--------------------|-----------------|---------------------|--------------------|--------------------|-------------------|------------|----------------------|
| XXS | 50M | 11M | 6 | 384 | 6 | 64 | 0.5M | 1.6×10^{-3} |
| XS | 77M | 26M | 8 | 512 | 8 | 64 | 0.5M | 1.4×10^{-3} |
| Small | 164M | 87M | 12 | 768 | 12 | 64 | 0.5M | 6.0×10^{-4} |
| Medium | 411M | 308M | 24 | 1024 | 16 | 64 | 0.5M | 3.0×10^{-4} |
| Large | 844M | 689M | 24 | 1536 | 16 | 96 | 0.5M | 2.5×10^{-4} |
| XL | 1.4B | 1.2B | 24 | 2048 | 16 | 128 | 1M | 4.0×10^{-4} |
| 2.8B | 2.8B | 2.6B | 32 | 2560 | 32 | 80 | 1M | 3.2×10^{-4} |
| 6.9B | 6.9B | 6.5B | 32 | 4096 | 32 | 128 | 2M | 2.4×10^{-4} |

Table 3: The model architecture for our suite of models. The 6 largest models follow similar architecture as models Small to 6.7B in the GPT-3 paper. The differences in the tables are due to minor calculation errors and typos in Table 2.1 of that paper. The n_{param} column has the total number parameters in each model while n_{ne} column has the number of parameters excluding the embedding and unembedding layers. Following [Kaplan et al., 2020], we use the number of non-embedding parameters in our scaling plots. We do not tie the weights in the embedding and unembedding layers.

B Scaling trends for FIM rate ablations

In Section 4.2, we see higher FIM rate improving the FIM performance of our models without impacting the original capabilities. This conclusion was based on the learning curves of HumanEval and light random span infilling pass rates measured with a small number of samples during pretraining. To further demonstrate this claim, we train a series of models for 50B tokens with FIM rates: 0, 0.25, 0.5, 0.75, 0.9, and 1.0. In Figure 12 and 13, we present the model scaling trends of perplexity and sampling evaluation when different FIM rates are used.

Again, we find that transforming a high fraction of training data into FIM does not result in a degradation in the original capabilities as measured by the test loss and HumanEval pass rate. The only noticeable degradation is observed in perplexity evaluation at 100% FIM rate. As for FIM capabilities, increasing the FIM rate yields a significant improvement on the infilling benchmarks and can change the slope of model scaling trends of pass rates. However, a high FIM rate does not lead to a commensurate reduction in FIM losses, which corroborates that perplexities do not always capture real world performance.

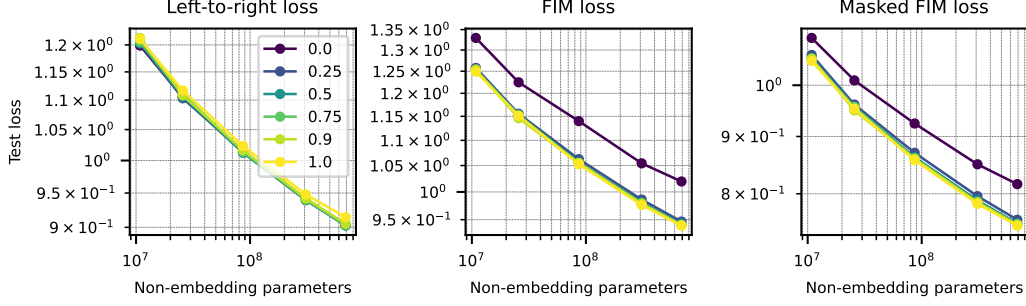


Figure 12: Comparison of model scaling trends of perplexity with varying FIM rates. Left-to-right loss does not have a noticeable degradation unless a FIM rate of 100% is used (left). We also find that the FIM losses are similar to one another when the model is trained with some FIM transformations (middle and right).

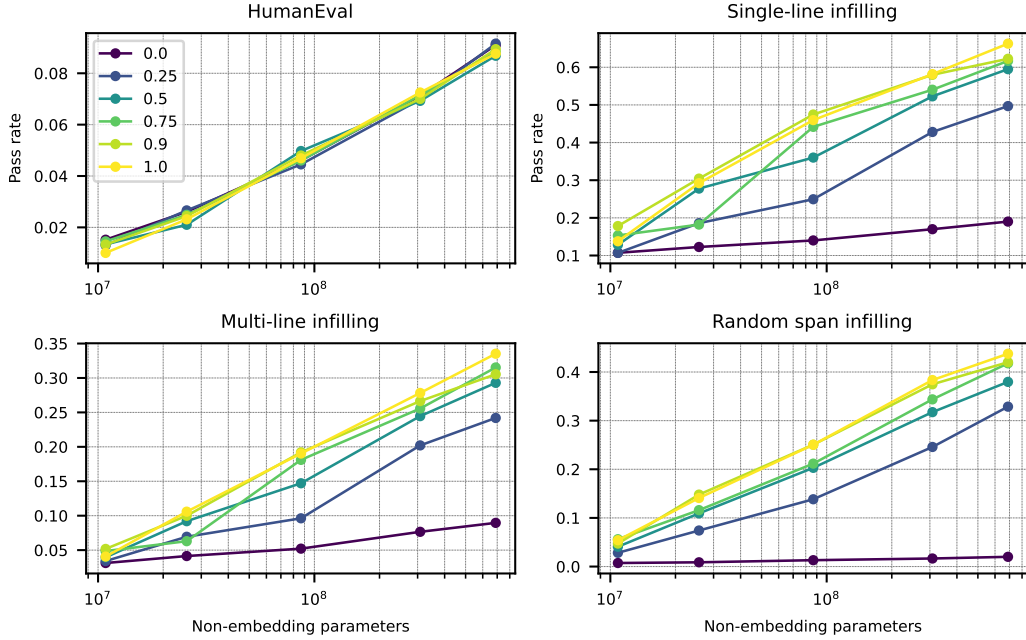


Figure 13: Comparison of model scaling trends of sampling evaluation with varying FIM rates. While increasing the FIM rate has no effects on HumanEval, it does result in consistent gains on the infilling benchmarks with no noticeable improvement after 90% FIM. At a first glance, it may seem counterintuitive that left-to-right models can solve a nontrivial number of problems in single- and multi-line benchmarks. This is not a bug, but a feature. We sample in SPM mode and some line-based infilling problems have empty or extraneous suffixes. To obtain these results, HumanEval was evaluated with temperature 0.8 and 500 samples per task to reduce variance. All infilling benchmarks, having much more problems than HumanEval, were evaluated with temperature 0.2 and 200 samples per task.

C Details of FIM implementation

When FIM is applied at the document level before packing, both character-level and token-level FIM is straightforward to implement. We simply choose two positions at random to break a document into three sections, and format them as a FIM document. Only the order of encoding and splitting changes as shown in the python pseudocode below:

```

def token_level_psm_fim(document: str, vocab: Vocab) -> List[int]:
    tokens = vocab.encode(document)
    prefix, middle, suffix = randomly_split(tokens)
    return [
        vocab.sentinel("prefix"), *prefix,
        vocab.sentinel("suffix"), *suffix,
        vocab.sentinel("middle"), *middle,
    ]

def character_level_psm_fim(document: str, vocab: Vocab) -> List[int]:
    prefix, middle, suffix = randomly_split(document)
    return [
        vocab.sentinel("prefix"), *vocab.encode(prefix),
        vocab.sentinel("suffix"), *vocab.encode(suffix),
        vocab.sentinel("middle"), *vocab.encode(middle),
    ]

```

In contrast, applying the transformation after packing and chunking as in context-level FIM can be somewhat tricky depending on the choice of middle span. As previously mentioned in Section 3, the input context to the model is first split around the `<EOT>` token so we get back individual documents. At this point, these documents are already tokenized, so applying FIM at the token level is straightforward.

To transform data in the character space for context-level FIM, the tokenized documents have to be decoded back into strings before FIM augmentation. Depending on the vocabulary, some care has to be given to ensure decoding does not introduce any spurious characters into training. For example, utf-8 characters are encoded as multiple tokens with a BPE vocabulary; they can result in fragments from chunking and fail to decode. To prevent unforeseen errors midway through training, we encourage checking for these fragments at the beginning or end of a context and removing them.

After the transformed documents are encoded and joined back, the resulting context can be longer or shorter than the original, unaugmented context for context- and character-level FIM. For this reason, we recommend to trim or pad the transformed context to the model context length.

D Details of SPM encoding

As mentioned in Section 3, in SPM we use the ordering `[suffix, prefix, middle]`. In this section, we briefly discuss the choices regarding the sentinel tokens in SPM mode. A natural choice of encoding for SPM data would be to use

`<SUF> ◦ Enc(suffix) ◦ <PRE> ◦ Enc(prefix) ◦ <MID> ◦ Enc(middle) ◦ <EOT>`. (SPM variant 1)

However, the encoding of SPM we use in this work is

`<PRE> ◦ <SUF> ◦ Enc(suffix) ◦ <MID> ◦ Enc(prefix) ◦ Enc(middle) ◦ <EOT>`. (SPM variant 2)

The reason that we do not use the former is that it creates a separation between PSM and SPM, which may result in less transfer between SPM and PSM. To understand, note that with the second variant SPM data occurs naturally as part of PSM training since when we split a document uniformly at random, sometimes the chosen prefix will be empty. This is the reason pure PSM runs achieve strong performance when evaluated in SPM mode as in Table 1.

Despite this, we note that the first SPM variant has its own advantages. In particular, it can be stronger in handling of subtokens at the end of prefix. Hence, the choice of which variant of SPM to use may depend on application in mind. As such, especially when training in pure SPM mode, it could be preferable to use the former simpler form. However, in this work, due to our emphasis on joint training of PSM and SPM and to maximize transfer between them, we opt for the second variant.

E Random span infilling benchmark

Fried et al. [2022] introduced the single-line and multi-line infilling benchmarks based on HumanEval which prove valuable for measuring FIM performance. One limitation of these benchmarks is that

the middle section is selected based on lines and does not capture more general use cases in the wild. We created a third infilling benchmark by choosing the middle span from two random positions in the canonical solution. In this section, we show some examples of these tasks so the reader can get a feel for the new benchmark. The goal is to predict the highlighted span.

```
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to
    each other than
    given threshold.
    """
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True

    return False
```

Here, for the model to pass, it needs to know that 1) the variable `distance` is not defined, 2) the prefix ends in a subtoken and not handling this will result in an indentation error, and 3) the completion has to stop in-line when the difference is calculated.

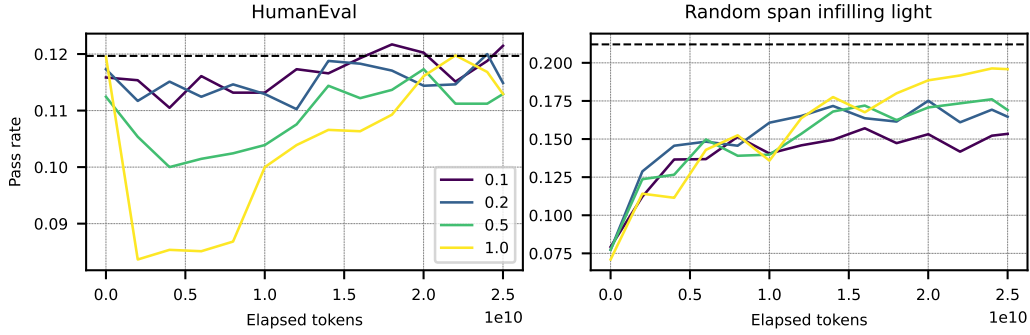
```
def rounded_avg(n, m):
    """You are given two positive integers n and m, and your task is to
    compute the
    average of the integers from n through m (including n and m).
    Round the answer to the nearest integer and convert that to binary.
    If n is greater than m, return -1.
    Example:
    rounded_avg(1, 5) => "0b11"
    rounded_avg(7, 5) => -1
    rounded_avg(10, 20) => "0b1111"
    rounded_avg(20, 33) => "0b11010"
    """
    if m < n:
        return -1
    summation = 0
    for i in range(n, m+1):
        summation += i
    return bin(round(summation/(m - n + 1)))
```

This is a slightly more difficult example where the missing section spans over multiple lines and ends in a subtoken which would break all previous works that use BPE encoding and token-based FIM. Use cases like this can happen in coding assistants when the user does not like the current implementation and quickly deletes an approximate span they want replaced by a code model.

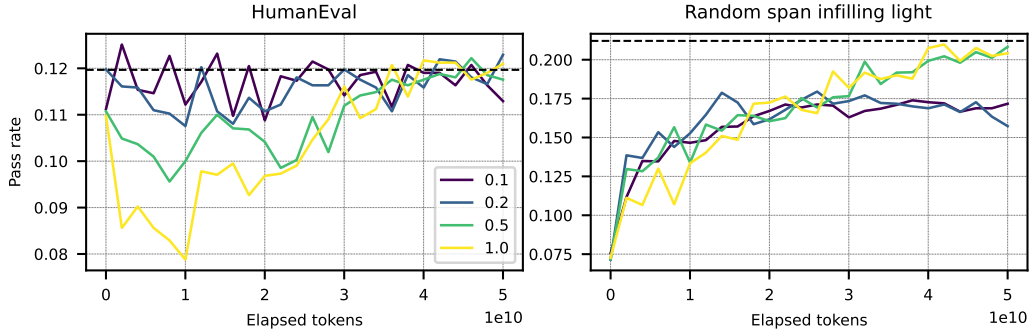
Because we create random span infilling tasks uniformly at random, this naturally captures problems of varying difficulties and corner cases that could happen in practice. We picked 10 random tasks per problem in HumanEval because 1640 tasks yielded a good balance between reducing evaluation variance and sampling time.

F Dynamics and learning curves of finetuning

To further build intuition about the results in Section 5, it is instructive to look at the dynamics of our infilling evaluation benchmarks during the finetuning. This is presented in Figure 14. We observe that the ordinary HumanEval degrades significantly at the beginning of finetuning, especially when using higher learning rates, but it catches up to similar levels as pretraining by the end of the training. On



(a) 25B tokens of FIM finetuning.



(b) 50B tokens of FIM finetuning.

Figure 14: The dynamics of HumanEval and random span infilling light during finetuning. The legend corresponds to the fraction of finetuning learning rate relative to the pretraining learning rate. The results here are with a FIM rate of 0.9 and we omit similar dynamics plots with a FIM rate of 0.5 for brevity.

the other hand, performance in random span infilling light starts out as zero as expected and slowly rises during finetuning.

G Top models comparison

In this section, we compare the performance of the current best infilling models on single-line, multi-line and random span infilling benchmarks. The results are reported in Table 4. We note that the numbers from InCoder in this table are self-reported numbers from the paper and was not independently evaluated in our framework. It is possible that minor differences in the implementation between our evaluation frameworks may result in slight discrepancies.

| Model | Single-line infilling | Multi-line infilling | Random span infilling |
|------------------|-----------------------|----------------------|-----------------------|
| FIM50 | 0.730 | 0.406 | 0.521 |
| FIM90 | 0.751 | 0.441 | 0.551 |
| INCODER | 0.690 | 0.386 | N/A |
| CODE-DAVINCI-002 | 0.916 | 0.699 | 0.742 |

Table 4: Comparison of our 6.9B parameter (6.5B non-embedding parameters) FIM model trained with a FIM rate of 50% and 90% for 100B tokens with the InCoder model of similar size (6.7B) and code-davinci-002, on the three main infilling benchmarks. All the FIM results are obtained in the SPM mode. We evaluated our models and code-davinci-002 using 100 samples our models and per task with a sampling temperature of 0.2.

H Qualitative evaluation

Previously, we measured the pass rates on coding benchmarks to assess the infilling capability. In this section, we qualitatively evaluate samples to understand the strengths and areas of improvement for FIM. Overall, we find that infilling works better on the code domain than language. However, as previously motivated, infilling is generally a more difficult task than just extending a prefix. We exemplify these challenges and show possible mitigations.

H.1 Successful infilling examples

FIM enables a model to process information from both before and after the point of generation. This unlocks new capabilities that previously required specialized models finetuned on specific tasks. For example, unlike Codex [Chen et al., 2021] that trained a separate docstring model, we now have a single model that can infer the import modules, function names, arguments, docstrings, definitions, and many more. We show one such example below that is impossible to complete unless the model can read the entire source code. This example is also interesting in that the prefix “from sym” and the suffix both contain subtokens, which are known to cause traditional language models trained without techniques like stochastic BPE [Provilkov et al., 2019] to fail.

```
from sympy import isprime

def largest_prime_factor(n):
    """
    Return the largest prime factor of n.
    """
    ans = 1
    for num in range(2, n + 1):
        if n % num == 0 and isprime(num):
            ans = num
    return ans
```

The benefits are not limited to coding. The model can adapt to the existing writing style and complete the passage in a natural way that takes the ending into consideration.

Dolphins are very intelligent animals. They are mammals and breathe air. They live in the sea and are related to whales and porpoises. Dolphins are very playful animals.

The commercial diver finally thought he'd snagged a big catch when he saw something white. But then he quickly realized it wasn't a fish -- he was wrangling an alligator.

Wikipedia is a free, web-based, collaborative, multilingual encyclopedia. It is overseen by the nonprofit Wikimedia Foundation. Wikipedia uses a collaborative software known as wiki that facilitates the creation and development of articles.

H.2 Limitations

Difficult prompts. Unlike completing text from the end, infilling needs to infer the missing span that connects the prefix to the suffix. When the suffix is completely unrelated, the model can generate very long middle sections. We consider this behavior as the model's attempt at coming up with a plausible trajectory that joins the ending. Because the context size is limited, the model usually fails to join. However, given that even people have trouble infilling some of these prompts in a short passage, this failure demonstrates how challenging of a task FIM can be.

Below, we show one such difficult prompt where the model typically fails to connect entirely or join in a seamless way. Even when the model writes a seemingly plausible middle section, the quality can often vary.

The dentist looked me in the eyes and said, "I'm going to have to take all of your teeth out."
I was stunned. I said, "All my teeth? Isn't there something else we could do?" He said, "No
, I'm afraid not."

No one can predict the future.

The Ottomans were defeated in World War I and the French were defeated at Waterloo.

Deciding when to stop. The model is trained to predict the <EOT> token when it thinks it has joined the suffix. Even when the prompts are seemingly straightforward, deciding when to end can still be a challenge in practice. Because there are many equally valid completions with varying lengths, the probability of outputting the <EOT> is discounted by other longer candidates and is often smaller than expected. This is further exacerbated by the fact that the terminal symbol can simply be missed due to sampling. This results in a behavior where the model does not seem to end in a timely manner and generates a valid, but spurious content in the middle. In the process, the model can choose to write its own ending to the prefix, effectively ignoring the given suffix.

Dogs are friendly animals.
Koalas are pleasant animals.
Monkeys are playful animals.
Whales are enormous animals.
Owls are wise animals.
Penguins are graceful animals.
Crocodiles are ferocious animals.

While the general problem of not knowing when to stop applies to normal left-to-right completion as well, this has not been as big a problem as infilling because there is no constraint to join the suffix.

Repetition. When the model fails to generate an <EOT> and copies the suffix, the model's ability to match patterns leads it to lock on and repeat the prompt indefinitely. Surprisingly, even large models are susceptible to this mode of failure. The example below ends with "the the heart," because the model has failed to generate the terminal symbol and is still in the middle of filling in the missing span which unfortunately will not stop.

The way is not in the sky. The way is in the heart.
The way is not in the sky. The way is in the heart.
The way is not in the sky. The way is in the heart.
The way is not in the sky. The way is in the the heart.

H.3 Mitigations

Like GPT-3 [Brown et al., 2020] where the performance depends on the quality of prompts, some of the failures in the earlier sections can be alleviated with prompt engineering. Namely, providing hints to constrain the output can dramatically improve the model's ability to generate the <EOT> token and connect to the suffix within a reasonable token budget as the model has a more concrete understanding of how long the middle section should be.

One such idea is to provide examples both in the beginning and the end with numbered items. This makes the model internally keep track of the position, pay attention to the desired prefix and suffix, and generally abstain from generating spurious content as shown below. Providing leading examples alone without any explicit cues can often worsen the problem because it does not resolve the ambiguity in whether the model should join to the beginning of the suffix or consider it as part of a new example.

1. Dogs are friendly animals.
2. Koalas are sleepy animals.
3. Lions are regal animals.

Section 1:

1. The way is not in the sky. The way is in the heart.
2. Peace comes from within. Do not seek it without.

Section 2:

It is important to note that the numbered few-shot prompting helps considerably but does not completely fix the problem, as the model can still accidentally start a new list of items.

In general, as the model can simply miss sampling the <EOT> token, we recommend generating multiple samples and preferring samples that end with <EOT>, as this increases the chance of choosing a sample that actually joins the ending. When multiple samples end in <EOT>, they can be reranked by the likelihood or other heuristics of interest. We call this EOT-aware best-of-n sampling.