Enabling Language Models to Fill in the Blanks

Chris Donahue
Stanford University

Mina Lee
Stanford University

Percy Liang
Stanford University

{cdonahue,minalee,pliang}@cs.stanford.edu

Abstract

We present a simple approach for *text infilling*, the task of predicting missing spans of text at any position in a document. While infilling could enable rich functionality especially for writing assistance tools, more attention has been devoted to language modeling—a special case of infilling where text is predicted at the end of a document. In this paper, we aim to extend the capabilities of language models (LMs) to the more general task of infilling. To this end, we train (or fine-tune) off-the-shelf LMs on sequences containing the concatenation of artificially-masked text and the text which was masked. We show that this approach, which we call *infilling by language modeling*, can enable LMs to infill entire sentences effectively on three different domains: short stories, scientific abstracts, and lyrics. Furthermore, we show that humans have difficulty identifying sentences infilled by our approach as machine-generated in the domain of short stories.

1 Introduction

Text infilling is the task of predicting missing spans of text which are consistent with the preceding and subsequent text.[1] Systems capable of infilling have the potential to enable rich applications such as assisting humans in editing or revising text (Shih et al., 2019), connecting fragmented ideas (AI21, 2019), and restoring ancient documents (Assael et al., 2019). Rather than targeting a particular application, our goal here is to provide a general, flexible, and simple infilling framework which can convincingly infill in a variety of domains.

A special case of infilling is language modeling: predicting text given preceding but not subsequent text.[2] Language models are (1) capable of generating remarkably coherent text (Zellers et al., 2019; See et al., 2019), (2) efficient at generating text, and (3) conceptually simple, but cannot infill effectively as they can only leverage context in a single direction (usually the past). On the other hand, strategies such as BERT (Devlin et al., 2019) and SpanBERT (Joshi et al., 2019) are able to infill using both preceding and subsequent text. However, their use of bidirectional attention limits their infilling capabilities to fixed-length spans. This is problematic as—for many applications—we may not know the length of a missing span *a priori*. Zhu et al. (2019) propose a method capable of infilling variable-length spans, but it uses a specialized architecture and hence cannot easily leverage large-scale pre-trained models.

In this work, we present infilling by language modeling (ILM), a simple framework which en-

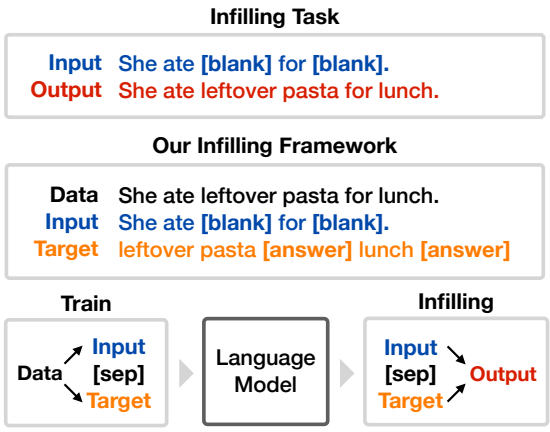

Figure 1: We consider the task of infilling, which takes incomplete text as input and outputs completed text. To tackle this task, our framework constructs training examples by masking random spans to generate pairs of inputs (text with blanks) and targets (answers for each blank). We then train unidirectional language models on the concatenation of each pair. Once trained, a model takes text input with blanks, predicts the answers, and then combines them to produce the output.

[1] Text infilling is a generalization of the *cloze* task (Taylor, 1953)—cloze historically refers to infilling individual words.

[2] In this paper, language modeling always refers to ordinary LMs, i.e., "unidirectional," "autoregressive," or "left-to-right."

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2492–2501
July 5 - 10, 2020.

ables LMs to infill variable-length spans while preserving their aforementioned benefits: generation quality, efficient sampling, and conceptual simplicity. Our framework involves a straightforward formulation of the infilling task which, as we demonstrate, can be learned effectively by existing LM architectures. As shown in Figure 1, our approach concatenates artificially-masked text with the text which was masked, and adopts a standard LM training (or fine-tuning) procedure on such examples. Once trained, infilling can be performed for a document with blanks by using the LM to generate text and then replacing the blanks with this text.

In addition to its conceptual simplicity, our experiments show that ILM enables off-the-shelf LMs to infill effectively. Furthermore, we find that infilling performance improves when starting from a large-scale pre-trained LM (as opposed to training from scratch), suggesting an additional benefit of using our model-agnostic framework compared to approaches which require specialized architectures.

We provide an interactive web demo of models trained under our framework. This demo can infill multiple variable-length spans with different granularities (e.g. words, n-grams, and sentences) on the domains of short stories, scientific abstracts, and song lyrics: https://chrisdonahue.com/ilm. All code, data, and trained models are available at https://github.com/chrisdonahue/ilm and also on the CodaLab platform at https://worksheets.codalab.org/worksheets/0x9987b5d9cce74cf4b2a5f84b54ee447b.

2 Problem Statement

The task of infilling is to take incomplete text $\tilde{x}$, containing one or more missing spans, and return completed text x. Let [blank] be a placeholder for a contiguous sequence (span) of one or more missing tokens. Then, incomplete text $\tilde{x}$ is a sequence of tokens some of which are [blank]. In order to map $\tilde{x}$ to x, an infilling strategy must specify both *how many* and *which* tokens to generate for each [blank]. Note that there may be many reasonable x for a given $\tilde{x}$. Hence, we are interested in learning a distribution $p(x \mid \tilde{x})$.

3 Infilling by Language Modeling

In this section, we describe our ILM framework. We first outline a simple reparametrization of the infilling task. Then, we define a procedure for automatically generating suitable training examples which can be fed to an off-the-shelf LM.

3.1 Formulation

Fedus et al. (2018) explore an infilling framework where LMs are trained on concatenations of $\tilde{x}$ and x, i.e., they use LMs to directly predict x given $\tilde{x}$. While their approach is effective at infilling individual words, it is somewhat redundant as the model must "predict" the unmasked text in $\tilde{x}$. Additionally, a model is not guaranteed to exactly reproduce the unmasked text.

Instead, we make the trivial observation that it suffices to predict only the missing spans y which will replace the [blank] tokens in $\tilde{x}$. We can then construct x by simply replacing [blank] tokens in $\tilde{x}$ with predicted spans y in a deterministic fashion. In order to handle multiple variable-length spans, we pose y as the concatenation of all missing spans separated by special [answer] tokens (one [answer] per [blank]) (Figure 1). We can thus cast infilling as learning $p(y \mid \tilde{x})$ without loss of generality.

3.2 Training

Given a corpus consisting of complete text examples, our framework first manufactures *infilling examples* and then trains an LM on these examples. To produce an infilling example for a given x, we first sample an $\tilde{x}$ from a stochastic function $\text{Mask}(x)$ which randomly replaces some number of spans in x with [blank] tokens. Then, we concatenate together the spans which were replaced—separated by [answer] tokens—to form a training target y. Finally, we construct the complete infilling example by concatenating $\tilde{x}$, [sep], and y (see Figure 2 for a complete example).

We train (or fine-tune) LMs on these infilling examples using standard LM training methodology, yielding models of the form $p_\theta(y \mid \tilde{x})$. Specifically, we train GPT-2 (Radford et al., 2019) off the shelf, but any LM can potentially be used.

This framework has several advantages. First, it incurs almost no computational overhead compared to language modeling. Specifically, if there are k missing spans in $\tilde{x}$, the concatenation of $\tilde{x}$ and y contains only $2k+1$ more tokens than x (one [blank] and one [answer] per missing span plus one [sep]). As k is usually small (averaging around 2 per example in our experiments), sequence lengths remain similar to those encountered for the same x during language modeling. In contrast, using LMs to directly predict x from $\tilde{x}$ as in Fedus et al. (2018) effectively doubles the sequence length of x.

This is particularly problematic when considering models like GPT-2 whose memory usage grows quadratically with sequence length. Second, our framework requires minimal change (three additional tokens) to an existing LM's vocabulary. Finally, because the entirety of $\tilde{x}$ is in the "past" when predicting y, the ILM framework combines the ability to attend to incorporate context on both sides of a blank with the simplicity of decoding from LMs.

4 Experimental Setup

We design our experiments to determine if training an off-the-shelf LM architecture with our ILM framework can produce effective infilling models for a variety of datasets. Specifically, we train on three datasets of different sizes and semantics (details in Appendix A): short STORIES (Mostafazadeh et al., 2016), CS paper ABSTRACTS, and song LYRICS.

4.1 Mask Function

A benefit of the ILM framework is that it can be trained to infill spans corrupted by arbitrary mask functions. Here, we explore a mask function which simultaneously trains models to infill different *granularities* of text; specifically, words, n-grams, sentences, paragraphs, and documents. By using a unique special token per granularity (e.g. [blank word]), this mask function offers users coarse but intuitive control over the length of the spans to be infilled.

We configure our mask function to mask each token in a given document with around 15% probability, echoing the configuration of Devlin et al. (2019). However, instead of masking individual tokens uniformly at random, we perform a pre-order traversal of the granularity hierarchy tree, randomly masking entire subtrees with 3% probability. For the datasets we consider, this results in a marginal token mask rate of about 15% (details in Appendix B).

While we train to infill several different granularities, we primarily evaluate and discuss the ability of our models to infill sentences for brevity. Quantitative results of our models on other granularities can be found in Appendix D, and granularity functionality can also be explored in our web demo.

4.2 Task and Model Configurations

For all experiments, we train the same architecture (GPT-2 "small") using the same hyperparameters (Appendix C) while varying the infilling strategy and dataset. In addition to our proposed ILM strategy for infilling, we consider three baseline strategies: (1) language modeling (LM; "infilling" based only on past context), (2) reverse language modeling (LM-Rev; "infilling" based only on future context), and (3) language modeling based on all available context (LM-All). LM-All simply concatenates x and $\tilde{x}$ together as in Fedus et al. (2018). LM-All represents arguably the simplest way one could conceive of infilling with LMs, but results in long sequence lengths. Training examples for all strategies are depicted in Figure 2.

Training Examples for Different Strategies

Data	She ate leftover pasta for lunch.
Masked	She ate **[blank]** for **[blank]**.
LM	She ate leftover pasta for lunch. **[end]**
LM-Rev	.lunch for leftover pasta ate She **[end]**
LM-All	She ate **[blank]** for **[blank]**. She ate leftover pasta for lunch. **[end]**
ILM	She ate **[blank]** for **[blank]**. **[sep]** leftover pasta **[answer]** lunch **[answer]**

Figure 2: Training examples for three baseline infilling strategies and ILM on a given artificially-masked sentence. For each strategy, we train the same architecture (GPT-2) on such examples. At both training and test time, examples are fed from left to right; anything to the left of a green target is available to the model as context when predicting the target. Precisely, LM only considers past context, and LM-Rev only considers future. LM-All considers all available context but uses long sequence lengths. Our proposed ILM considers all context while using fewer tokens.

For each strategy, we also vary whether training is initialized from the pre-trained GPT-2 model or from scratch. Despite discrepancies between the pre-training and our fine-tuning for most infilling strategies, *all* of the infilling experiments initialized from the pre-trained checkpoint performed better than their from-scratch counterparts. This indicates that ILM can effectively leverage large-scale language modeling pre-training to improve infilling performance. Henceforth, we will only discuss the models initialized from the pre-trained checkpoint, though we report quantitative performance for all models in Appendix D.

For the models trained on STORIES and ABSTRACTS, we trained models to convergence using early stopping based on the validation set perplexity (PPL) of each model computed only on the masked tokens. These models took about a day to reach

	STO	ABS	LYR	Length
LM	18.3	27.9	27.7	1.00
LM-Rev	27.1	46.5	34.3	1.00
LM-All	15.6	22.3	21.4	1.81
ILM	15.6	22.4	22.6	1.01

Table 1: Quantitative evaluation results. We report test set perplexity (PPL) on the sentence infilling task for different model configurations on all three datasets, as well as average length of all test set examples in tokens relative to that of the original sequence (lower is better for all columns). Our proposed ILM framework achieves better PPL than both LM and LM-Rev, implying that it is able to take advantage of both past and future context. ILM achieves similar PPL to LM-All with shorter sequence lengths (hence less memory).

their early stopping criteria on a single GPU. For the larger LYRICS dataset, we trained models for 2 epochs (about two days on a single GPU).

5 Quantitative Evaluation

We evaluate the quantitative performance of our models on the sentence infilling task by measuring PPL on test data.[3] In this setting, a sentence is selected at random and masked out, and we measure the likelihood assigned by a model to the masked sentence in the context of the rest of the document. Regardless of differences in the ordering and number of tokens that each strategy uses to represent a test example, PPL is always computed only for the span of tokens comprising the original sentence (e.g. green tokens in Figure 2).

Table 1 shows that across all datasets, ILM outperforms models which see only past or future context (LM and LM-Rev respectively), implying that our proposed framework is able to take advantage of bidirectional context despite using unidirectional models. Additionally, while one might expect LM-All to outperform ILM because its training examples more closely "resemble" those of standard LMs, ILM achieves similar performance to LM-All. This indicates that GPT-2 is able to effectively learn the "syntax" of ILM examples and achieve reasonable infilling performance with shorter sequences (and hence with much less memory usage).

We also observe that models trained via ILM perform similarly on the special case of language modeling compared to the models which were trained *only* on language modeling (Appendix D.1). This suggests that ILM does not just repurpose LMs to infill, but rather *extends* their capabilities while maintaining their original functionality.

[3]Overlap-based metrics such as BLEU score (Papineni et al., 2002) are not appropriate for evaluating infilling as there are many realistic infills that have no word-level overlap with the original, e.g., "a sandwich" instead of "leftover pasta."

6 Human Evaluation

In addition to our quantitative evaluation, we seek to evaluate the qualitative performance of ILM. To this end, we sample a story from the STORIES test set and randomly replace one of its five human-written sentences with a model output. Then, we task human annotators on Amazon Mechanical Turk with identifying which of the sentences in a story was machine-generated (details in Appendix E).

We compare our ILM model to three baseline infilling strategies: an LM (context beyond the replaced sentence was discarded), the best model (self-attention; SA) from Zhu et al. (2019), and the pre-trained BERT (base) model (Devlin et al., 2019). All approaches except for BERT were first fine-tuned on the STORIES dataset. To infill using BERT, we replace the tokens representing the original sentence with mask tokens, and then generate text by replacing mask tokens one at a time (conditioning on previously-generated tokens). While vocabulary differences make it is less useful to compare PPL for the SA and BERT baselines to our GPT-2-based strategies, we can still meaningfully compare them in this human evaluation setting.

For each approach we compute a *score*, which we define as the percentage of examples where the annotator did not correctly identify the machine-generated sentence. Therefore, a higher score implies a better (more natural, human-like) model. We collect 100 responses for each model and report the scores in Table 2, with qualitative examples in Figure 3 and Appendix E.

Of the four strategies, ILM achieves the highest score, implying that sentences infilled by ILM are harder for humans to recognize as fake than those produced by other strategies. Somewhat surprisingly, we observed that despite only observing past context the LM model performed better than BERT and SA. BERT may have performed poorly due to the intrinsic difficulty of finding convincing infills with a precise length in tokens. SA may have performed poorly because, unlike LM and ILM, it was not initialized from a large-scaled pre-trained LM.

	BERT	SA	LM	ILM
Score (%)	20	29	41	45

Table 2: Human evaluation results. We use BERT (Devlin et al., 2019), the best model from Zhu et al. (2019) (SA), and our LM and ILM models to replace random sentences in five-sentence stories from the STORIES test set. Then, we task humans with identifying which sentence of the five was generated by a machine. We report the *score* of each model: the percentage of infilled stories where the human failed to identify the machine-generated sentence. Our ILM model achieves a higher score than all of the other models. Note that the max score is effectively 80%, as a perfect model would cause annotators to randomly choose one of the five sentences.

Example Story with Masked Sentence

Patty was excited about having her friends over.
She had been working hard preparing the food.
[blank]
All of her friends arrived
and were seated at the table.
Patty had a great time with her friends.

BERT	favoritea ", Mary brightly said.
SA	She wasn't sure she had to go to the store.
LM	She went to check the tv.
ILM	Patty knew her friends wanted pizza.
Human	She also had the place looking spotless.

Figure 3: Example of a short story in our STORIES dataset with its third sentence masked, and sentences infilled by different models. The sentences generated by BERT and SA models are off-topic, the sentence generated by LM model is irrelevant to the future context, while the ones generated by ILM and Human successfully account for both previous and future context.

7 Related Work

Methodology. A number of systems have the capability to infill but have practical drawbacks. Many systems are unable to automatically determine span length, and thus, can only infill fixed-length spans (Fedus et al., 2018; Devlin et al., 2019; Yang et al., 2019; Joshi et al., 2019; Gu et al., 2019; Liu et al., 2019). Methods such as BERT present additional challenges during inference (Wang and Cho, 2019). Rudinger et al. (2015) frame narrative cloze as a generation task and employ language models, but they only consider one infill of a fixed length. Zhu et al. (2019); Shen et al. (2020) infill multiple variable-length sequences, but these approaches require the masked context to be iteratively updated and reprocessed to fill in blanks one a time. In contrast, our approach appends infilled text to the context and does not require reprocessing the entire input sequence for each blank. AI21 (2019) train an LM which can fill in the middle of a paragraph given the first and last sentences—our work generalizes to such capabilities.

Task. The cloze task (Taylor, 1953) evaluates language proficiency by asking systems to fill in randomly-deleted words by examining context. Cloze has been extended in the forms of discourse (Deyes, 1984) and narrative cloze (Chambers and Jurafsky, 2008), which remove phrases and narrative events respectively. Recently, cloze has been used not only for evaluation, but also to improve text generation quality (Fedus et al., 2018) and transfer learning (Devlin et al., 2019) (under the name "masked language modeling"). Text infilling can be thought of as generalizing the cloze task from single words to spans of unknown length. Raffel et al. (2019) explore infilling as a pre-training objective to improve downstream performance on inference tasks; our work focuses on generation.

Story generation. Recent work seeks to generate stories given a title and storyline (Yao et al., 2019), entities (Clark et al., 2018), premise (Fan et al., 2018), or surrounding context and rare words (Ippolito et al., 2019). Our work differs in that we aim to build systems capable of making predictions based only on text context, rather than aspects specific to stories (e.g. storyline).

8 Conclusion

We presented a simple strategy for the task of infilling which leverages language models. Our approach is capable of infilling sentences which humans have difficulty recognizing as machine-generated. Furthermore, we demonstrated that our infilling framework is effective when starting from large-scale pre-trained LMs, which may be useful in limited data settings. In future work, we plan to incorporate these features into co-creation systems which assist humans in the writing process. We hope that our work encourages more investigation of infilling, which may be a key missing element of current writing assistance tools.

Acknowledgments

This work was funded by DARPA CwC under ARO prime contract no. W911NF-15-1-0462. We thank all reviewers for their helpful comments.

References

AI21. 2019. HAIM: A modest step towards controllable text generation. *AI21 Labs Blog*.

Yannis Assael, Thea Sommerschield, and Jonathan Prag. 2019. Restoring ancient text using deep learning: a case study on greek epigraphy. *arXiv:1910.06262*.

N. Chambers and D. Jurafsky. 2008. Unsupervised learning of narrative event chains. In *Human Language Technology and Association for Computational Linguistics (HLT/ACL)*.

Elizabeth Clark, Yangfeng Ji, and Noah A Smith. 2018. Neural text generation in stories using entity representations as context. In *Association for Computational Linguistics: Human Language Technologies*.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Association for Computational Linguistics (ACL)*, pages 4171–4186.

T. Deyes. 1984. Towards an authentic 'discourse cloze'. *Applied Linguistics*, 5(2):128–137.

A. Fan, M. Lewis, and Y. Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

W. Fedus, I. Goodfellow, and A. M. Dai. 2018. Maskgan: Better text generation via filling in the. In *International Conference on Learning Representations (ICLR)*.

J. Gu, Q. Liu, and K. Cho. 2019. Insertion-based decoding with automatically inferred generation order. *arXiv preprint arXiv:1902.01370*.

D. Ippolito, D. Grangier, C. Callison-Burch, and D. Eck. 2019. Unsupervised hierarchical story infilling. In *NAACL Workshop on Narrative Understanding*, pages 37–43.

M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. 2019. SpanBERT: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

D. Liu, J. Fu, P. Liu, and J. Lv. 2019. TIGS: An inference algorithm for text infilling with gradient search. *arXiv preprint arXiv:1905.10752*.

N. Mostafazadeh, N. Chambers, X. He, D. Parikh, D. Batra, L. Vanderwende, P. Kohli, and J. Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *North American Association for Computational Linguistics (NAACL)*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *ACL*.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

R. Rudinger, P. Rastogi, F. Ferraro, and B. V. Durme. 2015. Script induction as language modeling. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Abigail See, Aneesh Pappu, Rohun Saxena, Akhila Yerukola, and Christopher D Manning. 2019. Do massively pretrained language models make better storytellers? *arXiv:1909.10705*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Tianxiao Shen, Victor Quach, Regina Barzilay, and Tommi Jaakkola. 2020. Blank language models. *arXiv:2002.03079*.

Y. Shih, W. Chang, and Y. Yang. 2019. XL-Editor: Post-editing sentences with xlnet. *arXiv preprint arXiv:1910.10479*.

W. L. Taylor. 1953. "Cloze procedure": A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433.

A. Wang and K. Cho. 2019. BERT has a mouth, and it must speak: BERT as a Markov random field language model. *arXiv preprint arXiv:1902.04094*.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew. 2019. HuggingFace's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

L. Yao, N. Peng, R. Weischedel, K. Knight, D. Zhao, and R. Yan. 2019. Plan-and-write: Towards better automatic storytelling. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. In *NeurIPS*.

W. Zhu, Z. Hu, and E. Xing. 2019. Text infilling. *arXiv preprint arXiv:1901.00158*.

A Datasets

- STORIES (100K examples, 5M words)
  Short stories from the ROCStories dataset (Mostafazadeh et al., 2016). Each story contains a title and five sentences.

- ABSTRACTS (200K examples, 30M words)
  Abstracts from CS papers on arXiv

- LYRICS (2M examples, 60M words)
  Song lyrics from `lyrics.com`

We experimented on multiple datasets to demonstrate that our framework was not custom tailored to a single domain. On the STORIES and ABSTRACTS datasets, we include metadata (story title, paper subject matter, etc.), as the first "paragraph" of the document. By providing these paragraphs (Appendix B), our infilling model implicitly learns to summarize (e.g. infill a title given a story), and do conditional generation (e.g. infill a story given a title). On the LYRICS dataset, infilling models may be especially helpful to humans; external aid in the form of rhyming dictionaries is already commonly employed in this domain.

To ensure that all experiments were trained on the same data, we removed infilling examples which would have exceeded our training sequence length of 256 tokens for the model with the longest sequence length (LM-All). This removed no examples from STORIES, a small fraction of examples from LYRICS, and a substantial number of examples from ABSTRACTS.

B Masking function

We design a mask function which takes the entire document and selectively masks several span granularities: words, n-grams, sentences, paragraphs, and entire documents. Accordingly, models trained via ILM on this masking function offer users the ability to specify the granularity of text to infill at a particular location. This allows users to have coarse but intuitive control over infilling length, so that multiple paragraphs are not generated when the user was expecting a single word.

Our masking function first constructs a tree of the training example (using the natural hierarchy of documents, paragraphs, sentences, and words). Then, using a pre-order tree traversal, each subtree is masked with 3% probability (or ignored if any of its ancestors are already masked). If the entire document (root node of the tree) is masked, then the infilling model's job is equivalent to that of a language model. If a word (leaf) is selected to be masked, 50% of the time we mask that individual word, otherwise we mask an n-gram of random length between 1 and $\min(8, \#\text{ words left in the sentence})$ words (inclusive). Note that a word may comprise multiple tokens, as GPT-2 uses sub-word tokenization (Sennrich et al., 2015). We chose the value of 3% as, for the datasets we considered, it resulted in a marginal token mask rate of around 15%, echoing the configuration of Devlin et al. (2019).

We add special tokens for each granularity to our model's vocabulary (e.g. [blank word]), so that the user may specify which granularity they would like the infilling model to produce. This functionality can be explored in our demo: `https://chrisdonahue.com/ilm`.

While we focus on this specific mask function in this paper, we structured the ILM codebase to allow users to train infilling models for completely different use cases. Users need only define a new mask function which takes complete documents and outputs lists of character-level spans representing the desired spans to be masked.

C Hyperparameters

We use early stopping based on the PPL of the model on infilling the masked token for the validation set. We train all models using the default fine-tuning parameters specified in the `transformers` library (Wolf et al., 2019), except that we use a batch size of 24 and a sequence length of 256.

Note that the most straightforward way of training an LM on ILM examples (Section 3.2) is to maximize the likelihood of the entire concatenated example: $\tilde{x}$, [sep], and y. This trains the model to predict tokens in $\tilde{x}$ even though such behavior is not necessary at inference time as $\tilde{x}$ will always be fully-specified. Nevertheless, we found that this additional supervision *improved* performance when evaluating model PPL of y. Conveniently, this is also the default behavior when adapting existing LM training code for use with ILM.

D Evaluation on language modeling and infilling other granularities

Our quantitative evaluation (Section 5) examined the sentence infilling performance of GPT-2 initialized from the large-scale pre-trained checkpoint

	STO	ABS	LYR
LM (scratch)	33.4	52.1	25.1
LM-Rev (scratch)	32.9	53.9	24.7
LM-All (scratch)	30.4	44.6	26.2
ILM (scratch)	30.8	45.3	30.6
LM	17.6	25.7	20.8
LM-Rev	25.1	36.7	23.7
LM-All	17.8	25.2	21.5
ILM	18.1	23.9	23.0

Table 3: Document infilling PPL (or language modeling) of ILM and baselines initialized either from scratch or from the pre-trained checkpoint across three datasets. Note that PPL of ILM is similar to LM, implying that our infilling strategy can reasonably maintain the ability to perform language modeling while extending the ability to infill.

	STO	ABS	LYR
LM (scratch)	34.0	52.8	28.9
LM-Rev (scratch)	34.9	59.3	30.4
LM-All (scratch)	27.0	46.2	24.3
ILM (scratch)	25.5	46.0	27.5
LM	17.5	25.5	23.9
LM-Rev	26.5	39.0	29.2
LM-All	15.1	24.4	19.3
ILM	14.9	23.5	20.2

Table 4: Mixture infilling PPL of all models (a mixture of all granularities).

after fine-tuning on different datasets and infilling strategies. Here, we report PPL for GPT-2 both initialized from scratch and from the pre-trained checkpoint for several other configurations: language modeling, a mixture of granularities, specific granularities, and language modeling.

D.1 Language modeling

In Table 3, we report PPL for "document infilling," which is equivalent to language modeling (because $\tilde{x}$ is always [blank document]). Because of how we structured our mask function (Appendix B), 3% of infilling examples consist of the entire document masked out, which results in the ability of our ILM framework to perform standard infilling. We see that performance of ILM is similar to that of LM on this task, even though ILM sees far fewer examples of language modeling compared to LM.

	STO	ABS	LYR
LM (scratch)	35.6	51.5	25.1
LM-Rev (scratch)	34.8	65.1	24.7
LM-All (scratch)	33.4	45.0	26.2
ILM (scratch)	34.3	45.3	30.6
LM	18.3	24.2	20.8
LM-Rev	26.5	42.8	23.7
LM-All	20.4	23.4	21.5
ILM	20.7	22.5	23.0

Table 5: Paragraph infilling PPL of all models.

	STO	ABS	LYR
LM (scratch)	36.0	65.4	33.5
LM-Rev (scratch)	35.1	92.2	35.8
LM-All (scratch)	27.1	53.8	27.1
ILM (scratch)	26.7	51.0	31.0
LM	18.3	27.9	27.7
LM-Rev	27.1	46.5	34.3
LM-All	15.6	22.3	21.4
ILM	15.6	22.4	22.6

Table 6: Sentence infilling PPL of all models.

D.2 Mixture of granularities

In Table 4, we report results for a mixture of granularities. Specifically, we run the same mask function we use for training (Appendix B) on our test data and evaluate PPL on the masked spans. This reflects general infilling ability across a wide variety of granularities (and hence lengths). Unlike our other quantitative evaluations, there may be multiple variable-length spans missing from each example in this evaluation. Results are similar to that of sentence infilling. Namely, that ILM outperforms LM and LM-Rev and is similar to LM-All despite using much less memory.

D.3 Individual granularities

In Tables 5 to 8 we report PPL values for infilling performance on paragraphs, sentences, n-grams, and words, respectively, across the three datasets.

For each granularity, we create one infilling example per document from the test set with exactly one masked span (randomly chosen from all spans of that granularity for that document). Then, we compute PPL only on the tokens which comprise the masked span, i.e., PPL is computed for all models on exactly the same set of tokens. Across all granularities, we observe that ILM outperforms

	STO	ABS	LYR
LM (scratch)	36.1	62.5	34.1
LM-Rev (scratch)	36.4	89.1	36.3
LM-All (scratch)	26.4	60.1	24.3
ILM (scratch)	23.1	49.5	26.3
LM	19.2	25.5	28.2
LM-Rev	26.6	45.0	34.8
LM-All	14.5	20.5	18.6
ILM	13.8	21.5	18.8

Table 7: N-gram infilling PPL of all models.

	STO	ABS	LYR
LM (scratch)	32.3	57.2	34.8
LM-Rev (scratch)	31.6	100.0	36.7
LM-All (scratch)	12.6	51.8	12.5
ILM (scratch)	9.2	37.9	12.2
LM	17.1	23.0	28.7
LM-Rev	24.1	45.0	35.1
LM-All	7.5	15.8	9.5
ILM	5.4	14.2	8.5

Table 8: Word infilling PPL of all models.

LM and LM-Rev and either outperforms or is comparable with LM-All while using less memory.

E Details on human evaluation

For human evaluation, we sampled 100 stories from the test set of the STORIES dataset. From each story, we masked out one sentence at a time, thereby resulting in 500 stories with masked sentences. Then we used these stories as context and tasked each model with infilling the masked sentence.

We compared 8 models in total. In addition to the four models reported in Section 6 (BERT, SA, LM, and ILM), we included the models which are initialized from scratch (as opposed to initialized from the large-scale pre-trained checkpoint) for exhaustive comparison. Furthermore, to filter out spam, we used a control model which always generates "This sentence was generated by a computer." Lastly, we included the original sentence from the dataset as a reference model (Human) to sanity check the max score is around 80%.

Each annotator was shown 8 stories, one from each model, and was asked to identify one of the five sentences generated by machine (see Figure 4 for an example). Among the 100 collected responses, we filtered out 5 responses whose annotation for the control model was wrong. The quantitative and qualitative results can be found in Table 9 and Figure 5, respectively. All model outputs and responses of human evaluation can be found at https://github.com/chrisdonahue/ilm.

	Score (%)
Control	0
BERT	20
SA	29
LM (scratch)	40
LM	41
ILM (scratch)	39
ILM	45
Human	78

Table 9: Human evaluation results.

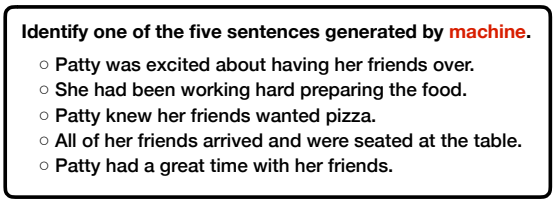

Figure 4: Example of a task and instruction for human evaluation on Amazon Mechanical Turk.

Example Story with Masked Sentence

Lily always loved to read.
She wondered sometimes,
what it would be like to write a book?
[blank]
Lily did well in the course, and during it,
wrote a short book.

BERT I held her hand and helped her sit.
SA Of her, but she didn't know her.
LM She practiced reading a lot every week.
ILM Finally, in middle school, her teacher introduced her to writing that.
Human She decided to take a course on fiction writing.

Example Story with Masked Sentence

Yesterday was Kelly's first concert.
She was nervous to get on stage.
[blank]
Kelly was then happy.
She couldn't wait to do it again.

BERT Or rather, what the next job would be now.
SA I was going out I was going to the beach.
LM I put on about thirty sugar cubes.
ILM The issues are getting so many people crazy.
Human I could never catch up and each week got worse.

Example Story with Masked Sentence

Yesterday was Kelly's first concert.
She was nervous to get on stage.
[blank]
Kelly was then happy.
She couldn't wait to do it again.

BERT Today was the first concert that she had to see every where.
SA She was going to go to the play.
LM When she went on stage she smoothly walked right past the audience.
ILM When she got on stage the band was amazing.
Human As soon as she got on the audience applauded.

Figure 5: Examples of sentence-level infills by different models.