

Structured Packing in LLM Training Improves Long Context Utilization

Konrad Staniszewski^{*1} Szymon Tworkowski^{*1} Sebastian Jaszczur¹ Yu Zhao² Henryk Michalewski³
Łukasz Kuciński¹ Piotr Miłoś^{*4}

Abstract

Recent developments in long-context large language models have attracted considerable attention. Yet, their real-world applications are often hindered by ineffective context information use. This work shows that structuring training data to increase semantic interdependence is an effective strategy for optimizing context utilization. To this end, we introduce Structured Packing for Long Context (SPLICE), a method for creating training examples by using information retrieval methods to collate mutually relevant documents into a single training context. We empirically validate SPLICE on large 3B and 7B models, showing perplexity improvements and better long-context utilization on downstream tasks. Remarkably, already relatively short fine-tuning with SPLICE is enough to attain these benefits. Additionally, the comprehensive study of SPLICE reveals intriguing transfer effects such as training on code data leading to perplexity improvements on text data.

1. Introduction

Large language models (LLMs) (Brown et al., 2020; Chowdhery et al., 2022; Lewkowycz et al., 2022; OpenAI, 2023b; Bai et al., 2023) have transformed the field of AI. Recently, the field has observed the rise of Long-Context Language Models (LCLMs) that promise to unveil novel and powerful capabilities (Anthropic, 2023; OpenAI, 2023a). However, their ability to process long context is not always as effective as one hopes for. Indeed, several studies have highlighted an important limitation: when processing prompts composed of multiple documents, LCLMs frequently encounter difficulties in accurately extracting relevant information (Tworkowski et al., 2023; Liu et al., 2023; Shi et al., 2023a). Additionally, they typically find it challenging to utilize information from the middle of

their inputs (Liu et al., 2023), even on simple synthetic retrieval tasks (Li et al., 2023a). Understanding these issues is vital for advancements in LCLM technologies and calls for systematic research.

In this work, we take a step towards better context utilization in LCLMs. We focus on training data, keeping other components, such as the architecture and training objectives, unchanged. The broad question is *how to organize training data to enhance long context capabilities?* Such perspective has received some attention recently (Levine et al., 2022; Chan et al., 2022; Shi et al., 2023b), but the problem is far from being solved. The central finding of this work is that *structuring training data to increase semantic interdependence is an effective strategy towards better long context utilization*. We achieve this by introducing and evaluating Structured Packing for Long Context (SPLICE), a method for creating training examples by using retrieval (e.g., BM25 or repository structure) to collate mutually relevant documents into a single training context.

We empirically validate SPLICE showing that fine-tuning of OpenLLaMA 3Bv2 and 7Bv2 (Geng & Liu, 2023) for only 2B–6B tokens already brings perplexity reduction. This reduction translates to substantial improvements in handling long-context information in downstream tasks that require retrieval and in-context learning. These tasks include TREC (Li & Roth, 2002; Hovy et al., 2001), DBpedia (Lehmann et al., 2015), Qasper (Shaham et al., 2022; Dasigi et al., 2021), HotPotQA (Yang et al., 2018), and the lost-in-the-middle benchmark (Liu et al., 2023). We perform a comprehensive study of the design choices and properties of SPLICE, showing that the acquired long-context capabilities transfer between modalities, such as code and text.

Our contributions can be summarized as follows:

- We comprehensively show that structuring training data is a viable way of improving the long context utilization of LLMs. To this end, we introduce SPLICE, a method for creating training examples by using retrieval to collate mutually relevant documents into a single training context.
- We fine-tune OpenLLaMA 3Bv2 and OpenLLaMA 7Bv2 (Geng & Liu, 2023) using SPLICE, showing that it improves perplexity on long context evaluation,

¹University of Warsaw ²University of Edinburgh ³Google DeepMind ⁴Polish Academy of Sciences. Correspondence to: Konrad Staniszewski <ks.staniszewski@uw.edu.pl>.

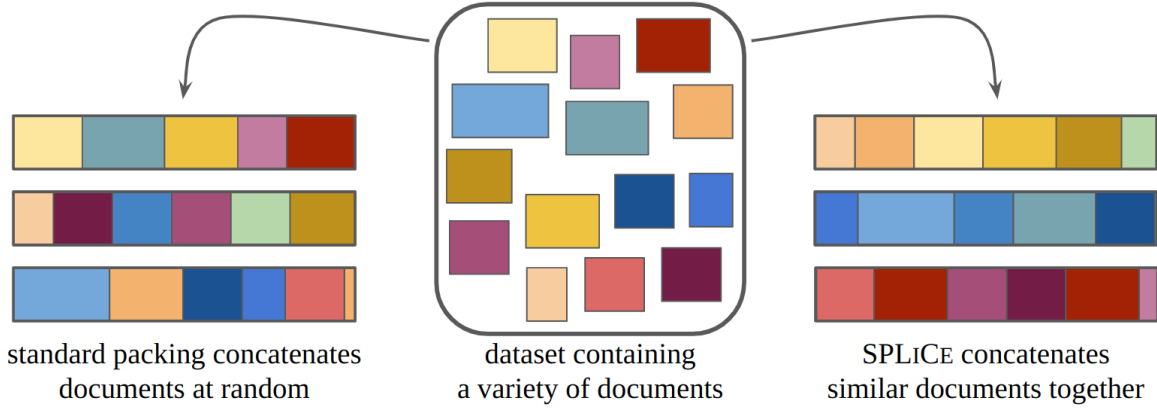


Figure 1: Training examples generated by the standard sampling procedure vs proposed by SPLICE. Similar colors indicate related documents, which could be found using a retrieval method (e.g., BM25 or contriver) or metadata (e.g., repository structure for code).

in-context learning ability, and ability to retrieve relevant data from the context.

- We present a thorough analysis of the design choices behind SPLICE, such as the number of retrieved documents and the order in which documents are merged into a training example.

2. Method

SPLICE is a method for constructing training examples that enhances the LLMs long-context utilization. This translates to improved performance on a range of tasks like in-context learning, question answering, in-context information retrieval, and long context language modeling. We make the source code available to ensure the reproducibility of our results. (For this anonymous submission we share the zip file with our code.)

Rationale and intuitions Capturing long-range dependencies is believed to enhance language modeling and retrieval-augmentation (Borgeaud et al., 2022). However, it is an open question how to achieve such benefits in pre-training or fine-tuning. The primary difficulty comes from the fact that long-range dependencies are rare in training data (de Vries, 2023) and diminish with distance. It is thus unlikely that a model will learn to utilize long context without more explicit guidance.

Recent studies indicate that structuring data, i.e., going beyond the i.i.d. paradigm, might be beneficial or even necessary to achieve good long-context performance. (Levine et al., 2022) develops a theory showing that the trained model establishes stronger dependencies between text segments in the same training example. (Chan et al., 2022) indicates that specific distributional properties of the training data have a significant impact on the model’s in-context performance. Finally, (Shi et al., 2023b) shows that training

Algorithm 1 SPLICE training example construction

Input:

D : document corpus
 k : breadth hyper-parameter
 L : maximum length of returned training example
 RETRIEVE: retrieval method to use, e.g., BM25
 ORDER: ordering method, e.g., id, or random shuffle

Output: training example consisting of concatenated documents

```

 $d_r \sim D$  {Sample the root document}
 $D = D \setminus \{d_r\}$ 
 $C = [d_r]$ 
 $Q = \text{empty queue}$ 
 $Q.\text{PUSH}(d_r)$ 
while  $Q \neq \emptyset$  and  $\text{len}(C) \leq L$  do
     $d = Q.\text{POP}()$ 
     $d_1, \dots, d_k = \text{RETRIEVE}(d, k)$ 
    {Retrieve top- $k$  most similar documents to  $d$  using a
    selected method, e.g., BM25}
    for each  $d_i$  in  $d_1, \dots, d_k$  do
        if  $d_i \in D$  then
            {RETRIEVE uses a precomputed index, and may
            return documents that are already in  $C$ }
             $C = C.\text{APPEND}(d_i)$  {Append  $d_i$  to  $C$ }
             $Q.\text{PUSH}(d_i)$ 
             $D = D \setminus \{d_i\}$ 
        end if
    end for
end while
return  $\text{TRIM}(\text{ORDER}(C), L)$ 
    
```

on structured data improves the performance of LLMs on long-context tasks.

SPLICE follows these intuitions. Namely, it constructs training examples by collating mutually relevant documents to increase the dependency density, thus allowing the model to learn to utilize long context. We provide additional insights into the distributional properties of SPLICE created data at the end of this section and in Appendix K.

Structured Packing for Long Context (SPLICE) We propose SPLICE, a method that creates training examples by building a tree of documents, see Algorithm 1 and Figure 1 for a general overview.

SPLICE starts by picking a random document from the dataset to create a single-node tree and continues in a breadth-first manner, each time appending top- k similar documents from the corpus. The final sequence consists of the tree flattened according to a certain tree traversal strategy.

The hyperparameter k introduces flexibility and allows to interpolate between modes of retrieval. Indeed, for $k = 1$ SPLICE creates a path of related examples simulating a long document. For larger values of k , SPLICE produces examples that are similar to the ones used by retrieval augmented models, e.g., (Borgeaud et al., 2022). Note that the dataset must be reasonably curated for SPLICE to work efficiently. For example, a low duplication rate incentivizes the model to utilize long context beyond simple copying.

Many possible retrieval methods can be used with SPLICE (RETRIEVE function in Algorithm 1). In our experiments, we test the following:

- **SPLICE_{REPO}**: based on additional meta-information about the data, that is the repository structure of the code (REPO): we concatenate files using a depth-first search algorithm on the directory structure, that is files from the same directory are grouped together. A similar method has been pioneered by (Wu et al., 2022) and proposed in (Shi et al., 2023b) as an interesting future direction.
- **SPLICE_{BM25}**: based on BM25 (Robertson & Zaragoza, 2009; Bassani, 2023), a standard retrieval method that uses a bag-of-words approach to rank documents based on their similarity to a query.
- **SPLICE_{CONT}**: based on Contriever-MSMARCO (CONT) (Izacard et al., 2022), a recently proposed retrieval method that uses a transformer model to rank documents based on their similarity to a query.

SPLICE computational efficiency Given the dataset sizes used in training LLMs, the computational efficiency plays

a non-trivial role. SPLICE_{REPO} is the fastest and easy to implement but requires additional directory structure, i.e., is not applicable to general web data. SPLICE_{BM25} requires quadratic time index and needs dataset chunking to be applicable to large datasets, see Section 3.3. SPLICE_{CONT} requires calculating embeddings for each document and retrieval based on the cosine similarity. The latter step can be done efficiently using fast approximate inner-product search, e.g., Faiss (Johnson et al., 2017).

Baseline The standard approach, commonly used in LLM training pipelines, is to randomly sample documents from the dataset and concatenate them to make training examples of a desired context. This is known as example packing (Brown et al., 2020).

Burstiness SPLICE conjecturally falls into the framework presented in (Chan et al., 2022). This paper shows that the distributional properties of the training data affect the in-context capabilities of transformer models. In particular, it indicates the importance of "burstiness", i.e., a flatter frequency distribution with a relatively higher mass on the rare, long-tail tokens appearing in a sequence. In Appendix K, we show that SPLICE increases the burstiness of the training data (measured in terms of Zipf's coefficient of token frequency) compared to the baseline.

3. Experiments with Medium-Scale Models

In this section, we conduct a comprehensive examination of the impact of document packing on long-context performance, using medium-scale models (see Section 4 for large models results). We demonstrate that structuring data with SPLICE is a generally applicable approach to improve long context performance. We study the performance in Section 3.2 and design choices of SPLICE in Section 3.3.

3.1. Experimental Setup

In this part, we employ 270M models and the following protocol. Initially, we train with the 2K context length on 6.3B tokens from RedPajama (TogetherComputer, 2023). Subsequently, we continue training using 1B tokens with an extended context length of 32K on a mixture of the original RedPajama data (TogetherComputer, 2023) and long context data created using SPLICE/BASELINE. We employ the Focused Transformer (FoT) objective (Tworkowski et al., 2023) for context extension (unless stated otherwise). This approach is motivated by practical factors, viz. training with short context length expedites the process, while context scaling can be achieved by finetuning on a relatively small amount of tokens, as demonstrated by (Chen et al., 2023; Tworkowski et al., 2023). Loosely inspired by (Ouyang et al., 2022; Rozière et al., 2023), in the latter phase, long context data (i.e., prepared with SPLICE) constitutes half of

the mixture. This aims to prevent the model from overfitting to artificially created long documents.

In the evaluation, we measure perplexity on held-out portions of the arXiv (Azerbayev et al., 2022) and StarCoder (Li et al., 2023b) datasets employing a context length of 32K. The selection of these datasets is motivated by their inclusion of documents that can benefit from long-context information as demonstrated in (Chen et al., 2023) and (Li et al., 2023b). For example, functions are often re-utilized, and similar terms are employed across papers. We exclude documents with fewer than 32K tokens and truncate those exceeding this length.

We refer to Appendix A for a detailed description of the model architecture. For information regarding the training and evaluation data, see Appendix G.

3.2. Experimental Results

In this section, we compare SPLICE against BASELINE. We test two retrieval approaches for SPLICE: BM25 (Robertson & Zaragoza, 2009; Bassani, 2023), and Contriever-MSMARCO (Izacard et al., 2022)) denoted as SPLICE_{BM25} and SPLICE_{CONT} respectively. In SPLICE_{BM25} retrieval takes into account the whole content of the document, whereas in SPLICE_{CONT} we use the first 512 tokens and Faiss (Johnson et al., 2017) for fast approximate inner-product search (see Appendix H for details). By SPLICE_{REPO}, we denote the approach using the directory structure of code repositories. We also test our method with 64K context length and report the results in Table 19 in Appendix J. In those experiments, we use $k = 1$; see Section 3.3 for more details.

The results are presented in Table 1, Table 2, and Table 3, from which we draw the following conclusions.

Structuring data improves performance. We observe (see Table 1) that all variants of SPLICE outperform BASELINE, often significantly, regardless of the training and eval dataset. This indicates that the structure of the data is important for long-context training.

The positive effects transfer between domains. We observe (see Table 1) positive effects when training on code and evaluating on arXiv and when training on web data and evaluating on code. This indicates that the positive effects of SPLICE transfer between domains. We conjecture that better data mixtures could bring further benefits.

Retrieval method is of secondary importance. We observe (see Table 1) that the choice of retrieval method has a minor impact on the results. Typically, SPLICE_{BM25} is slightly better than SPLICE_{CONT}. On the code data, SPLICE_{REPO} is on par with SPLICE_{BM25}. We recall that SPLICE_{REPO} can be applied only to code data, whereas

SPLICE_{BM25} and SPLICE_{CONT} are more general. In practice, we often observe that SPLICE_{CONT} is faster than SPLICE_{BM25} due to the use of Faiss.

SPLICE works with various context extension results

Our main experiments use the Focused Transformer (FoT) approach (Tworkowski et al., 2023) for context extension. FoT extends the context only in a couple of attention layers and does not change the positional embeddings of the remaining layers. This allows for computationally efficient long-context fine-tuning. To show that our method works more generally, we have also evaluated it using more standard approaches. That is, we checked the effectiveness of the method when context extension is done in all layers, and parameters of Rotary Positional Encodings are either adjusted as in CodeLlama (Rozière et al., 2023), adjusted with YaRN (Peng et al., 2023), or left without changes. Table 3 shows the results.

Training is stable. We prepared three subsets of the C code dataset and ran the tested methods on each of them. In Table 2, we report the mean perplexity and its standard deviation. We observe that the differences between the subsets are minimal, which indicates training stability and confirms the statistical significance of our results.

3.3. Design Choices Analysis

Choice of k As already mentioned in the previous section, the choice of the retrieval method (BM25 or Contriver) plays a minor role. In Appendix C.1, we study the role of k . We first observe that its choice has a minor impact on the results, with $k = 1$ being the best choice. We leave for further work a more detailed analysis of this hyperparameter.

Document ordering We discovered that the ordering of the documents collected by the retrieval procedure might be a subtle issue. We consider three choices of ORDER function in Algorithm 1: identity (do not permute the documents), reverse (reverse the order of documents) and a random shuffle. In Appendix C.1, we present results for 270M models showing negligible differences. However, we observed differences for large models, elaborated in Section 4.2.4, in which random shuffling is the best choice for the CodeLlama context extension method.

Corpus chunking Performing retrieval from the whole corpus might be cumbersome due to its scale. However, we find that chunking the corpus into smaller parts still brings improvements. In particular, for the experiments with English Wikipedia and BM25 presented in Table 1, we used a manageable size of 2GB.

Table 1: Perplexity (improvement over BASELINE) for tested methods. We train a 270M parameter model with 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and long-context data C, C#, Python, Wikipedia, StackExchange prepared using a method of choice (SPLICE_{BM25}, SPLICE_{CONT}, SPLICE_{REPO}, BASELINE). BASELINE denotes organizing long-context data in the same way as RedPajama. SPLICE beats the BASELINE often by a large margin. The variants of SPLICE perform similarly, with SPLICE_{BM25} being slightly better. For detailed results, see Appendix B.

Long Context Data	Method	arXiv	Code				Code & arXiv
			Haskell	Python	CUDA	All	
C	SPLICE _{BM25}	5.46 (.09)	3.20 (.17)	2.81 (.12)	2.22 (.11)	2.94 (.13)	3.10 (.13)
	SPLICE _{CONT}	5.48 (.07)	3.22 (.15)	2.82 (.11)	2.23 (.10)	2.96 (.11)	3.11 (.12)
	SPLICE _{REPO}	5.47 (.08)	3.22 (.15)	2.83 (.10)	2.24 (.09)	2.96 (.11)	3.12 (.11)
	BASELINE	5.55	3.37	2.93	2.33	3.07	3.23
C#	SPLICE _{BM25}	5.52 (.13)	3.33 (.25)	2.90 (.17)	2.46 (.19)	3.11 (.20)	3.26 (.20)
	SPLICE _{CONT}	5.53 (.12)	3.35 (.23)	2.91 (.16)	2.48 (.17)	3.12 (.19)	3.27 (.19)
	SPLICE _{REPO}	5.53 (.12)	3.35 (.23)	2.91 (.16)	2.49 (.16)	3.12 (.19)	3.27 (.19)
	BASELINE	5.65	3.58	3.07	2.65	3.31	3.46
Python	SPLICE _{BM25}	5.47 (.10)	3.25 (.21)	2.53 (.09)	2.41 (.15)	3.02 (.15)	3.17 (.15)
	SPLICE _{CONT}	5.49 (.08)	3.28 (.18)	2.53 (.09)	2.43 (.13)	3.03 (.14)	3.19 (.13)
	SPLICE _{REPO}	5.48 (.09)	3.27 (.19)	2.54 (.08)	2.44 (.12)	3.03 (.14)	3.18 (.14)
	BASELINE	5.57	3.46	2.62	2.56	3.17	3.32
Wikipedia	SPLICE _{BM25}	5.64 (.09)	3.82 (.15)	3.26 (.11)	2.87 (.13)	3.55 (.13)	3.68 (.13)
	SPLICE _{CONT}	5.65 (.08)	3.87 (.10)	3.30 (.07)	2.92 (.08)	3.59 (.09)	3.72 (.09)
	BASELINE	5.73	3.97	3.37	3.00	3.68	3.81
StackExchange	SPLICE _{BM25}	5.07 (.07)	3.88 (.06)	3.32 (.04)	2.89 (.05)	3.60 (.05)	3.69 (.05)
	SPLICE _{CONT}	5.09 (.05)	3.91 (.03)	3.35 (.01)	2.93 (.01)	3.63 (.02)	3.73 (.01)
	BASELINE	5.14	3.94	3.36	2.94	3.65	3.74

Table 2: Perplexity for training on a 50/50 data mixture of RedPajama and C code. We perform three trainings using different training datasets and report the mean and standard deviation.

Long Context Data	Method	arXiv	Code		Code & arXiv
			Python	All	
C	SPLICE _{BM25}	5.463 $\pm$.002	2.810 $\pm$.002	2.942 $\pm$.005	3.100 $\pm$.004
	SPLICE _{CONT}	5.477 $\pm$.005	2.824 $\pm$.001	2.957 $\pm$.006	3.115 $\pm$.006
	SPLICE _{REPO}	5.474 $\pm$.007	2.827 $\pm$.006	2.958 $\pm$.009	3.115 $\pm$.009
	BASELINE	5.550 $\pm$.002	2.931 $\pm$.008	3.073 $\pm$.006	3.228 $\pm$.005

Table 3: Perplexity (improvement over BASELINE) for training on a 50/50 data mixture of RedPajama, and C#. We check that SPLICE brings improvements when fine-tuning for the longer context (16K) using the method of CodeLlama (Rozière et al., 2023), YaRN (Peng et al., 2023), or left without changes (Naive), as opposed to FoT used in the other experiments. For details see Table 13.

RoPe scale method	Training data	Method	arXiv	Code				Code & arXiv
				Haskell	Python	CUDA	All	
Naive	C#	SPLICE _{BM25}	6.25 (.08)	4.84 (.19)	3.55 (.11)	2.84 (.12)	3.72 (.13)	3.88 (.12)
		SPLICE _{REPO}	6.25 (.08)	4.87 (.16)	3.56 (.10)	2.85 (.11)	3.74 (.11)	3.89 (.11)
		BASELINE	6.33	5.03	3.66	2.96	3.85	4.00
CodeLlama	C#	SPLICE _{BM25}	5.74 (.02)	4.28 (.09)	3.22 (.05)	2.53 (.05)	3.34 (.06)	3.49 (.06)
		SPLICE _{REPO}	5.74 (.02)	4.28 (.09)	3.22 (.05)	2.54 (.04)	3.35 (.05)	3.50 (.05)
		BASELINE	5.76	4.37	3.27	2.58	3.40	3.55
YaRN	C#	SPLICE _{BM25}	5.77 (.02)	4.32 (.10)	3.24 (.05)	2.55 (.06)	3.37 (.07)	3.52 (.06)
		SPLICE _{REPO}	5.77 (.02)	4.32 (.10)	3.24 (.05)	2.56 (.05)	3.38 (.06)	3.53 (.05)
		BASELINE	5.79	4.42	3.29	2.61	3.44	3.58

4. Large-Scale Models

In this section, we show that SPLICE can improve the long context performance of large-scale language models. To this end, we use 3B and 7B parameter models and tasks that test in-context learning (Section 4.2.1), question answering (Section 4.2.2), and in-context information retrieval capabilities (Section 4.2.3). We provide perplexity results in Appendix E.

We also emphasize that those improvements occur during a relatively short, compared to pre-training, fine-tuning. To be more precise, 3B models were tuned on 5.4B tokens, whereas 7B models were tuned on 2B tokens. All of the models were pre-trained on 1T tokens.

4.1. Experimental Setup

For 3B experiments, we continue training OpenLLaMA 3B v2 (Geng & Liu, 2023; Geng, 2023) on a 50/50 mixture of RedPajama prepared in the standard way and C prepared using SPLICE_{BM25}. For 7B ones, we continue the training of OpenLLaMA 7B v2 on 75/25/25 mixture of RedPajama (50) prepared in the standard way, StackExchange (25) and C (25) prepared using SPLICE_{BM25}. StackExchange is part of RedPajama (TogetherComputer, 2023), and C data come from StarCoder (Li et al., 2023b).

We train with 32K context length using 5.4B tokens for 3B models and 2B for 7B models (recall that OpenLLaMA models have the 2K context length). We use a batch size of 256K tokens per step and the learning rate of $1.5e-5$ with linear warmup and cosine decay, following (Geng & Liu, 2023).

Regarding the context extensions method we use FoT (Tworkowski et al., 2023), and the CodeLlama (CL) (Rozière et al., 2023) approach. More explicitly, we test **six models**:

$$\{3B \text{ FoT}, 7B \text{ FoT}, 7B \text{ CL}\} \times \{\text{SPLICE}, \text{BASELINE}\},$$

where BASELINE denotes the standard data preparation method. Hyperparameter details are located in Appendix A.

4.2. Experimental Results

We present perplexity measurements in Appendix E. In particular, following (Anthropic, 2023), we examine perplexity improvements in the function of the token position in the document. Our findings reveal that SPLICE models demonstrate superior perplexity across all distances. Notably, the improvement in perplexity is more pronounced for tokens positioned later in the document, indicating enhanced language modeling capabilities.

For the remainder of this section, we focus on testing the in-context learning, question answering and information

retrieval abilities on our models using downstream tasks.

4.2.1. IN-CONTEXT LEARNING

To evaluate the improvements of in-context learning abilities, we use TREC (Li & Roth, 2002; Hovy et al., 2001) and DBpedia (Lehmann et al., 2015), which are text classification tasks. For TREC, to test the long context utilization, we vary the number of in-context examples in $\{190, 380, 780, 1560\}$ (which correspond roughly to $\{4, 8, 16, 32\}$ K context length). We sample these examples multiple times and measure classification accuracy on test set. In Table 4, we observe that SPLICE improves the average performance across different context lengths and model sizes. We follow a very similar protocol for DBpedia, see Table 5, confirming that SPLICE improves the context utilization. In Appendix I, we study the distribution of improvements with respect to the choice of the in-context examples, showing the domination of SPLICE.

In Table 4 and Table 5, we report mean improvement Δ along with 95% bootstrap confidence intervals. See Appendix I for related histograms.

Table 4: Average classification performance on TREC (Li & Roth, 2002; Hovy et al., 2001). For TREC, we average across 50 sets of in-context examples for 3B models and 10 sets for 7B models. We use Δ [confidence interval] to denote mean improvement and its 95% bootstrap confidence intervals (see Appendix I).

TREC				
Model	Context (#examples)	BASELINE	SPLICE	Δ [conf interv]
3B _{FoT}	32K (1560)	73.9	79.3	5.4 [4.7, 6.2]
	16K (780)	68.9	76.9	8.0 [6.9, 9.3]
	8K (380)	66.5	75.8	9.3 [8.1, 10.8]
	4K (190)	65.1	72.8	7.7 [6.5, 9.0]
7B _{FoT}	32K (1560)	75.6	79.4	3.8 [2.1, 5.1]
	16K (780)	74.0	79.0	5.0 [3.4, 6.0]
	8K (380)	72.1	76.2	4.1 [2.9, 5.7]
	4K (190)	66.9	69.7	2.8 [1.5, 4.2]
7B _{CL}	32K (1560)	75.3	76.6	1.3 [0.8, 1.8]
	16K (780)	81.4	82.5	1.1 [0.2, 1.6]
	8K (380)	76.6	76.5	-0.1 [-1.0, 1.1]
	4K (190)	68.1	69.1	1.0 [-0.4, 2.9]

4.2.2. QUESTION ANSWERING

We also show that SPLICE models have improved question answering capabilities. To this end, we use Qasper (Dasigi et al., 2021; Shaham et al., 2022) and HotPotQA (Yang et al., 2018). Qasper contains questions regarding research papers provided as a part of the input context. In Table 7, we

Table 5: Average performance on DBpedia (Lehmann et al., 2015). We average results across 40 sets of in-context examples, with Δ [confidence interval] denoting mean improvement and its 95% bootstrap confidence interval. Due to the size of the evaluation dataset for each set of in-context examples, we subsample a subset of 500 elements of the evaluation set.

DBpedia				
Model	Context (#examples)	BASELINE	SPLICE	Δ [conf intv]
3B _{FoT}	32K (380)	82.9	85.9	3.0 [2.6, 3.6]
	16K (190)	79.1	82.0	2.9 [2.5, 3.4]
7B _{FoT}	32K (380)	82.9	84.9	2.0 [1.5, 2.4]
	16K (190)	83.6	85.6	2.0 [1.5, 2.5]
7B _{CL}	32K (380)	95.1	95.6	0.5 [0.3, 0.6]
	16K (190)	96.2	96.4	0.2 [0.0, 0.3]

Table 6: Average performance on HotPotQA (Yang et al., 2018). We average results across 7 sets of in-context examples, with Δ [confidence interval] denoting mean improvement and its 95% bootstrap confidence intervals.

HotPotQA				
Model	Context (#examples)	BASELINE	SPLICE	Δ [conf intv]
3B _{FoT}	20K (10)	29.7	29.6	-0.1 [-0.3, 0.1]
7B _{FoT}	20K (10)	26.0	27.3	1.3 [1.2, 1.5]
7B _{CL}	20K (10)	31.0	31.2	0.2 [0.1, 0.6]

observe that SPLICE improves the two-shot performance. Similarly, we observe improvements on HotPotQA, see Table 6. We stress that in both cases, the results measure in-context capabilities as neither of the datasets was used during training.

4.2.3. KEY RETRIEVAL PERFORMANCE

The key retrieval task introduced in (Liu et al., 2023) has become a routine diagnostic tool to study context capabilities. Specifically, the model is presented with a list of key-value pairs, which are 128-bit UUIDs, and is tasked to retrieve the value for a given key. Even though very simple, the task proved to be challenging for many open-source language models, see Appendix D for the task details.

In Figure 2, we present the key retrieval performance of 7B models trained with CodeLlama (Rozière et al., 2023) context extension method. We note that despite relatively short tuning, SPLICE significantly helps on hard-to-retrieve positions. We provide the results for FoT models in Appendix D.

Table 7: Two-shot F1 performance on the validation subset of Qasper (Dasigi et al., 2021). For Qasper, we use the implementation from Language Model Evaluation Harness (Gao et al., 2021). Note that in the 3B model case, despite using SPLICE for code data only, we still have improvements in non-code tasks.

QASPER			
Model	Context length (#examples)	BASELINE	SPLICE
3B _{FoT}	32K (2)	22.1	22.8
7B _{FoT}	32K (2)	22.8	23.1
7B _{CL}	32K (2)	29.0	29.7

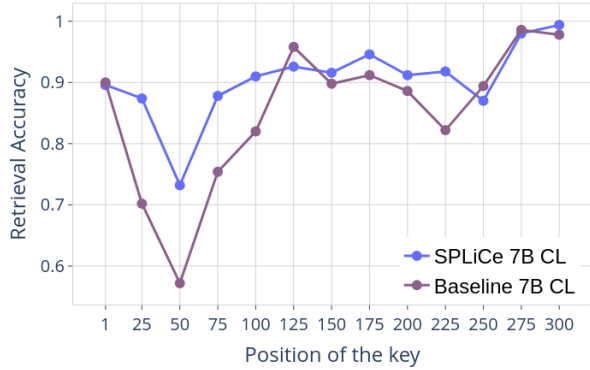


Figure 2: Key-value retrieval performance on a dictionary of 300 key-value pairs ($\approx 24K$ tokens). The 7B CL model trained with SPLICE achieves much higher accuracy on hard-to-retrieve positions. The details about this task can be found in Appendix D. Each position was evaluated using 500 examples.

4.2.4. SHUFFLING

In our experiments with large 7B CodeLlama models, we have noted slight improvements in handling long inputs when shuffling documents inside the context, that is, setting `Order` in the Algorithm 1 to be a random shuffle. We provide a comparison on TREC in Table 17.

5. Related Work

There is an increasing number of works aiming to study the role of data in LLM training in detail. For instance, (Levine et al., 2022) developed a theory and demonstrated empirically that incorporating non-adjacent but semantically related sentences in training examples leads to better sentence embeddings and improves open-domain question-answering performance. Another work (Gu et al., 2023) introduced a pretraining framework grounded on the idea that text documents often include intrinsic tasks. They showed

that this approach substantially boosts in-context learning. Additionally, there is existing work on training long-context language models using repository-level code data, such as (Wu et al., 2022). A recent work (Chan et al., 2022) identifies the training data’s distributional properties that affect transformer models’ in-context capabilities. Similarly, (Han et al., 2023) constructs small-scale data using an iterative gradient approach and shows that such data improve in-context performance.

Our methodology diverges from these works in several key ways. Firstly, we focus on the document-level context during the training phase, as opposed to sentence-level (Levine et al., 2022) or paragraph-level (Gu et al., 2023) granularity. We demonstrate the efficacy of this approach in large-scale language modeling, specifically with OpenLLaMA 3B. Secondly, we construct a tree structure of related documents through BM25/Contriever-MSMARCO retrieval and linearize this structure to form long-context examples. This allows for greater control over example coherence compared to relying solely on natural data structures like repository-level code. The gradient-based method presented in (Han et al., 2023) can be broadly associated with retrieval used in our work; however, it differs in scale and granularity.

Concurrently to our research, (Shi et al., 2023b) showed a similar method for preparing the training data. However, in our work, we focus on tuning the models for long context, up to 32K tokens, whereas (Shi et al., 2023b) focuses on training the models from scratch, with context length up to 8K tokens. In addition to that, we provide ablations regarding context extension methods.

6. Limitations and Future Work

We show that structuring the training data is a viable way of improving the model’s performance. The presented method can be viewed as a general framework for organizing the documents into training examples. This opens multiple further research avenues.

Degree of relevance In this work, SPLICE constructed training examples by concatenating most matching documents (according to BM25/CONT/REPO). However, recent results of (Tworkowski et al., 2023) show that introducing unrelated data to the context may help the model learn better representations. We leave the study of how the choice of retriever (in particular, the ratio of related and unrelated documents) affects performance for future work. Note that as BM25 is a syntactic retriever, it is not guaranteed that the documents returned by it are semantically related. Moreover, we have not tuned Contriever-MSMARCO, and it was provided only with a 512 token long prefix of a document.

In order to prepare the training data, SPLICE uses each document exactly once (we mask out each used document

for further retrieval). However, it is possible that allowing some high-quality documents to occur more than once may be beneficial.

Retrieval granularity Another avenue for future work is to study the granularity of the pieces from which the training examples are constructed. In this work, we focus on the document-level granularity. However, it is possible to construct training examples from smaller pieces, such as paragraphs or sentences.

Scaling Our method requires further studies to understand its scaling properties beyond the range presented in the paper.

Other context extension methods In most cases, our models were trained for a long context using the methodology presented in (Tworkowski et al., 2023). Additionally, we tested three popular context extension methods on a medium scale (Naive, YaRN, and CodeLlama) and tuned a large 7B model using the CodeLlama approach, preliminarily confirming the applicability of SPLICE. However, we leave more detailed studies of the design choices of SPLICE with other context extension methods to future work.

Other data sources One of the approaches to training long-context language models is to use conversational data (Li et al., 2023a). This is complementary to our method. SPLICE can utilize data that already exists in vast quantities and can be easily applied to different types of text (like code, Wikipedia articles, StackExchange questions, answers, etc.) to further increase the number of long-context examples. We leave researching how SPLICE integrates with other methods for preparing the long-context data as future work.

Data curation Using highly correlated samples has the potential to result in training instability. However, we noted no performance degradation during our experiments. We leave the study of how SPLICE integrates with different data types for the future.

7. Conclusions

In this work, we present SPLICE, a novel method of constructing training examples for long-context language models, which utilizes BM25/Contriever-MSMARCO to find relevant documents and feed them to the model in a structured manner. We show that SPLICE improves the perplexity of the language modeling task and performance on downstream tasks. We further show that SPLICE can be used to improve long-context utilization of large-scale models using only short fine-tuning. We believe that our work indicates multiple interesting research directions for improving the performance of long-context language models with structured data.

8. Impact Statement

This paper presents work whose goal is to improve the long context utilization ability of language models through structuring training data. It bears standard risks associated with advancing the field of LLMs (such as misuse and generation of malicious/misleading content). For a detailed overview, we refer to (Bender et al., 2021; Weidinger et al., 2021).

Acknowledgments

We gratefully acknowledge that our research was supported with Cloud TPUs from Google’s TPU Research Cloud (TRC). We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH, CI TASK) for providing computer facilities and support within computational grant numbers PLG/2023/016265. Piotr Miłoś was supported by National Science Center Poland under the grant agreement 2019/35/O/ST6/03464.

References

- Anthropic. Model card and evaluations for claude models. Technical report, Anthropic, 2023. URL <https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf>.
- Azerbayev, Z., Piotrowski, B., and Avigad, J. Proofnet: A benchmark for autoformalizing and formally proving undergraduate-level mathematics problems. In *Advances in Neural Information Processing Systems 35, 2nd MATH-AI Workshop at NeurIPS’22*, 2022. URL <https://mathai2022.github.io/papers/20.pdf>.
- Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., Hui, B., Ji, L., Li, M., Lin, J., Lin, R., Liu, D., Liu, G., Lu, C., Lu, K., Ma, J., Men, R., Ren, X., Ren, X., Tan, C., Tan, S., Tu, J., Wang, P., Wang, S., Wang, W., Wu, S., Xu, B., Xu, J., Yang, A., Yang, H., Yang, J., Yang, S., Yao, Y., Yu, B., Yuan, H., Yuan, Z., Zhang, J., Zhang, X., Zhang, Y., Zhang, Z., Zhou, C., Zhou, J., Zhou, X., and Zhu, T. Qwen technical report, 2023.
- Bassani, E. retriV: A Python Search Engine for the Common Man, May 2023. URL <https://github.com/AmenRa/retriV>.
- Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT ’21, pp. 610–623, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445922. URL <https://doi.org/10.1145/3442188.3445922>.
- Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022. URL <https://proceedings.mlr.press/v162/borgeaud22a.html>.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL <https://arxiv.org/abs/2005.14165>.
- Chan, S., Santoro, A., Lampinen, A. K., Wang, J., Singh, A., Richemond, P. H., McClelland, J. L., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. In *NeurIPS*, 2022.
- Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation, 2023.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. Palm: Scaling language modeling with pathways, 2022.
- Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers, 2021.

- de Vries, H. In the long (context) run, 2023. URL <https://www.harmdevries.com/post/context-length>. Accessed: 2023-09-28.
- Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, sep 2021. URL <https://doi.org/10.5281/zenodo.5371628>.
- Geng, X. EasyLM: A simple and scalable training framework for large language models, March 2023. URL <https://github.com/young-geng/EasyLM>.
- Geng, X. and Liu, H. Openllama: An open reproduction of llama, May 2023. URL https://github.com/openlm-research/open_llama.
- Gu, Y., Dong, L., Wei, F., and Huang, M. Pre-training to learn in context. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4849–4870, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.267. URL <https://aclanthology.org/2023.acl-long.267>.
- Han, X., Simig, D., Mihaylov, T., Tsvetkov, Y., Celikyilmaz, A., and Wang, T. Understanding in-context learning via supportive pretraining data, 2023.
- Hovy, E., Gerber, L., Hermjakob, U., Lin, C.-Y., and Ravichandran, D. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research*, 2001. URL <https://www.aclweb.org/anthology/H01-1069>.
- Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022, 2022. URL <https://openreview.net/forum?id=jKN1pXi7b0>.
- Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus, 2017.
- Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7 (3):535–547, 2019.
- Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. Dbpedia - A large-scale, multi-lingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015. doi: 10.3233/SW-140134. URL <https://doi.org/10.3233/SW-140134>.
- Levine, Y., Wies, N., Jannai, D., Navon, D., Hoshen, Y., and Shashua, A. The inductive bias of in-context learning: Rethinking pretraining example design. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL <https://openreview.net/forum?id=lnEaqbTJIRz>.
- Lewkowycz, A., Andreassen, A. J., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V. V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving quantitative reasoning problems with language models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL <https://openreview.net/forum?id=IFXTZERXdm7>.
- Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J. E., Stoica, I., Ma, X., and Zhang, H. How long can open-source llms truly promise on context length?, June 2023a. URL <https://lmsys.org/blog/2023-06-29-longchat>.
- Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., V. R. M., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Moustafa-Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you! *CoRR*, abs/2305.06161, 2023b. doi: 10.48550/arXiv.2305.06161. URL <https://doi.org/10.48550/arXiv.2305.06161>.
- Li, X. and Roth, D. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL <https://www.aclweb.org/anthology/C02-1150>.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *CoRR*, abs/2307.03172, 2023. doi: 10.48550/arXiv.2307.03172. URL <https://doi.org/10.48550/arXiv.2307.03172>.
- OpenAI. New models and developer products. OpenAI Blog, 2023a. URL <https://blog.salesforceairesearch.com/xgen-7b/>.

- OpenAI. Gpt-4 technical report, 2023b.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models, 2023.
- Robertson, S. E. and Zaragoza, H. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. doi: 10.1561/15000000019.
- Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code llama: Open foundation models for code, 2023.
- Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., and Levy, O. Scrolls: Standardized comparison over long language sequences, 2022.
- Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E. H., Schärli, N., and Zhou, D. Large language models can be easily distracted by irrelevant context. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023a. URL <https://proceedings.mlr.press/v202/shi23a.html>.
- Shi, W., Min, S., Lomeli, M., Zhou, C., Li, M., James, R., Lin, X. V., Smith, N. A., Zettlemoyer, L., Yih, S., and Lewis, M. In-context pretraining: Language modeling beyond document boundaries, 2023b.
- Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL <https://arxiv.org/abs/2104.09864>.
- TogetherComputer. Redpajama: An open source recipe to reproduce llama training dataset, April 2023. URL <https://github.com/togethercomputer/RedPajama-Data>.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- Tworowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Milos, P. Focused transformer: Contrastive training for context scaling. *NeurIPS 2023*, 2023.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL <http://arxiv.org/abs/1706.03762>.
- Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., Kenton, Z., Brown, S., Hawkins, W., Stepleton, T., Biles, C., Birhane, A., Haas, J., Rimell, L., Hendricks, L. A., Isaac, W., Legassick, S., Irving, G., and Gabriel, I. Ethical and social risks of harm from language models, 2021.
- Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL <https://openreview.net/forum?id=TrjbxzRcnf->.
- Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL <https://doi.org/10.18653/v1/d18-1259>.

A. Architecture

The architecture of our models is based on LLaMA (Touvron et al., 2023), and the architectural details can be found in Table 8. Briefly speaking, our architecture is similar to the one introduced in (Vaswani et al., 2017) with a few standard changes. First, we use only the decoder without the encoder part. Secondly, we perform RMSNorm before the input of both the attention and feed-forward modules. Thirdly, we use the LLaMA FeedForward module. Additionally, we use Rotary Position Embedding (Su et al., 2021). For context extension, we use Focused Transformer (FoT)(Tworkowski et al., 2023), CodeLlama (Rozière et al., 2023) (CL) and YaRN (Peng et al., 2023). Table 9 presents the details about both standard and long-context pretraining.

Table 8: Architecture details. Focused Transformer context extension is applied in continued pretraining for 32K context and evaluation.

Parameter/Model Size	270M	3B	7B
Vocab Size	32000	32000	32000
Embedding Size	1024	3200	4096
Num Attention Layers	12	26	32
Num Attention Heads	8	32	32
Head Size	128	100	128
MLP Hidden Size	4096	8640	11008
FoT Context Extension Layers	[6, 9]	[6, 12, 18]	[8, 16, 24]

Table 9: Training details. We pretrain a custom 270M parameter model and take a pretrained 3B/7B parameter OpenLLaMAv2 model (Geng, 2023). Subscript denotes that parameter was specific for a context extension method with FoT referring to (Tworkowski et al., 2023) and no-FoT to other methods (Naive, YaRN (Peng et al., 2023), CodeLlama (Rozière et al., 2023)).

Stage	Parameter/Model Size	270M	3B	7B
Pretraining	Context	2K	2K	2K
	Tokens	6.3B	1T	1T
Long Context Pretraining	Context	32K _{FoT} , 16K _{no-FoT}	32K	32K
	Batch Size	128 _{FoT} , 16 _{no-FoT}	128	128 _{FoT} , 32 _{no-FoT}
	Start Learning Rate	5e-5	1.5e-5	1.5e-5
	End Learning Rate	5e-6	1.5e-6	1.5e-6
	Warmup Steps	250	1000	1000 _{FoT} , 200 _{no-FoT}
	Tokens	1B	5.4B	2B

B. Detailed Results For Medium-Size Models

In this section, we extend results presented in Section 3. The training methodology is described in Section 3.1. Details about the number of tokens used for evaluation can be found in Table 18.

Tables 10 and 11 show the results of training 270M parameter model for 32K context on a 50/50 mixture of RedPajama data (organized in a standard way) and code data organized using a specified method. Table 10 contains detailed results from training on C# and Python. Table 11 contains results on C averaged across three different subsets of C (for details about the construction of those subsets, see Appendix G). Both tables show that SPLICE outperforms the BASELINE by a significant margin.

The main advantage of SPLICE_{BM25}/SPLICE_{CONT} over the SPLICE_{REPO} approach is that it can be used also for non-structured data. Table 12 shows the detailed results of applying the SPLICE on non-code data. Note that training on non-code data allows us to improve the model perplexity on the arXiv dataset in comparison to the model trained on code.

Table 13 considers models trained with YaRN (Peng et al., 2023), CodeLlama (Rozière et al., 2023) and Naive (no adjustment to RoPE) context extension methods. Table 14 shows that a simple artificial extension of example length via random concatenation of documents does not help.

Table 10: Perplexity results comparing different ways of organizing the same data. All runs started from the same 270M model with 2048 context and were trained for 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and code organized in the mentioned ways. For details about training please refer to Section 3.1.

Altered Train Data: C#				
Eval/Method	SPLICE _{BM25}	SPLICE _{CONT}	BASILINE	SPLICE _{REPO}
ArXiv	5.52	5.53	5.65	5.53
C	2.39	2.40	2.50	2.40
C++	2.60	2.61	2.74	2.62
CUDA	2.46	2.48	2.65	2.49
C#	1.82	1.82	1.90	1.82
Common Lisp	3.41	3.44	3.72	3.42
Dart	2.14	2.16	2.31	2.16
Emacs Lisp	8.41	8.46	8.85	8.41
Erlang	2.80	2.80	2.95	2.81
Fortran	3.68	3.71	4.05	3.72
Go	1.94	1.95	2.06	1.96
Groovy	3.01	3.03	3.25	3.04
Haskell	3.33	3.35	3.58	3.35
Java	2.09	2.10	2.22	2.10
Pascal	3.61	3.62	3.80	3.60
Python	2.90	2.91	3.07	2.91
Mean	3.26	3.27	3.46	3.27

Altered Train Data: Python				
Eval/Method	SPLICE _{BM25}	SPLICE _{CONT}	BASILINE	SPLICE _{REPO}
ArXiv	5.47	5.49	5.57	5.48
C	2.38	2.39	2.45	2.39
C++	2.61	2.63	2.71	2.63
CUDA	2.41	2.43	2.56	2.44
C#	1.98	1.99	2.06	2.00
Common Lisp	3.23	3.28	3.47	3.27
Dart	2.15	2.17	2.27	2.17
Emacs Lisp	7.94	7.98	8.28	7.93
Erlang	2.70	2.71	2.82	2.71
Fortran	3.42	3.46	3.72	3.46
Go	1.95	1.96	2.03	1.96
Groovy	2.97	2.99	3.13	2.99
Haskell	3.25	3.28	3.46	3.27
Java	2.12	2.12	2.20	2.13
Pascal	3.57	3.58	3.73	3.58
Python	2.53	2.53	2.62	2.54
Mean	3.17	3.19	3.32	3.18

Table 11: To check the statistical significance of our results, we prepare three subsets of C (details in Appendix G and train the models on a 50/50 mixture of RedPajama data (organized in the standard way) and C data organized using one of the methods. Note that the standard deviation is much lower than the perplexity improvements from using SPLICE.

Eval/Method	Altered Train Data: C			
	SPLICE _{BM25}	SPLICE _{CONT}	BASELINE	SPLICE _{REPO}
ArXiv	5.463 $\pm$ 0.002	5.477 $\pm$ 0.005	5.550 $\pm$ 0.002	5.474 $\pm$ 0.007
C	2.126 $\pm$ 0.020	2.134 $\pm$ 0.020	2.173 $\pm$ 0.023	2.135 $\pm$ 0.020
C++	2.396 $\pm$ 0.004	2.403 $\pm$ 0.002	2.467 $\pm$ 0.001	2.403 $\pm$ 0.006
CUDA	2.219 $\pm$ 0.002	2.235 $\pm$ 0.005	2.330 $\pm$ 0.009	2.239 $\pm$ 0.004
C#	1.939 $\pm$ 0.000	1.948 $\pm$ 0.001	2.016 $\pm$ 0.004	1.949 $\pm$ 0.002
Common Lisp	2.994 $\pm$ 0.072	3.042 $\pm$ 0.091	3.195 $\pm$ 0.078	3.043 $\pm$ 0.102
Dart	2.141 $\pm$ 0.002	2.155 $\pm$ 0.004	2.268 $\pm$ 0.009	2.156 $\pm$ 0.002
Emacs Lisp	7.857 $\pm$ 0.017	7.851 $\pm$ 0.020	8.098 $\pm$ 0.027	7.840 $\pm$ 0.019
Erlang	2.665 $\pm$ 0.002	2.680 $\pm$ 0.003	2.769 $\pm$ 0.007	2.676 $\pm$ 0.003
Fortran	3.306 $\pm$ 0.010	3.335 $\pm$ 0.010	3.554 $\pm$ 0.010	3.333 $\pm$ 0.012
Go	1.910 $\pm$ 0.001	1.924 $\pm$ 0.007	1.999 $\pm$ 0.002	1.924 $\pm$ 0.006
Groovy	3.009 $\pm$ 0.001	3.026 $\pm$ 0.006	3.147 $\pm$ 0.013	3.025 $\pm$ 0.007
Haskell	3.198 $\pm$ 0.001	3.221 $\pm$ 0.001	3.371 $\pm$ 0.008	3.220 $\pm$ 0.008
Java	2.075 $\pm$ 0.002	2.086 $\pm$ 0.001	2.161 $\pm$ 0.006	2.085 $\pm$ 0.005
Pascal	3.492 $\pm$ 0.026	3.496 $\pm$ 0.019	3.622 $\pm$ 0.021	3.513 $\pm$ 0.014
Python	2.810 $\pm$ 0.002	2.824 $\pm$ 0.001	2.931 $\pm$ 0.008	2.827 $\pm$ 0.006
Mean	3.100 $\pm$ 0.004	3.115 $\pm$ 0.006	3.228 $\pm$ 0.005	3.115 $\pm$ 0.009

Table 12: Perplexity results comparing different ways of organizing the same data. All runs started from the same 270M model with 2048 context and were trained for 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and other data organized using one of the methods. For details about training, please refer to Section 3.1. Note that the model trained with SPLiCE on StackExchange outperforms the one trained on code on arXiv evaluation, showing the benefits of SPLiCE’s applicability to non-code data.

Altered Train Data: StackExchange			
Eval/Method	SPLiCE _{BM25}	SPLiCE _{CONT}	BASELINE
ArXiv	5.07	5.09	5.14
C	2.68	2.69	2.70
C++	3.02	3.04	3.06
CUDA	2.89	2.93	2.94
C#	2.27	2.28	2.29
Common Lisp	4.02	4.06	4.08
Dart	2.58	2.60	2.61
Emacs Lisp	9.55	9.67	9.69
Erlang	3.13	3.16	3.18
Fortran	4.28	4.34	4.38
Go	2.24	2.25	2.27
Groovy	3.62	3.66	3.68
Haskell	3.88	3.91	3.94
Java	2.43	2.45	2.45
Pascal	4.08	4.11	4.14
Python	3.32	3.35	3.36
Mean	3.69	3.73	3.74

Altered Train Data: Wikipedia			
Eval/Method	SPLiCE _{BM25}	SPLiCE _{CONT}	BASELINE
ArXiv	5.64	5.65	5.73
C	2.65	2.67	2.71
C++	2.98	3.01	3.07
CUDA	2.87	2.92	3.00
C#	2.22	2.24	2.29
Common Lisp	3.87	3.96	4.08
Dart	2.51	2.55	2.61
Emacs Lisp	9.38	9.45	9.63
Erlang	3.13	3.16	3.23
Fortran	4.23	4.32	4.49
Go	2.18	2.21	2.26
Groovy	3.49	3.55	3.67
Haskell	3.82	3.87	3.97
Java	2.39	2.41	2.46
Pascal	4.32	4.23	4.40
Python	3.26	3.30	3.37
Mean	3.68	3.72	3.81

Structured Packing in LLM Training Improves Long Context Utilization

Table 13: Perplexity results comparing different ways of organizing the same data for non-FoT models. All runs started from the same 270M model with 2048 context and were trained for 16K context on a 50/50 mixture of RedPajama (organized in a standard way) and C# code is organized in one of three ways.

Altered Train Data: C#			
Context 16K: CodeLlama			
Eval/Method	SPLICE _{BM25}	BASELINE	SPLICE _{REPO}
ArXiv	5.74	5.76	5.74
C	2.66	2.70	2.66
C++	2.79	2.83	2.79
CUDA	2.53	2.58	2.54
C#	1.91	1.93	1.91
Common Lisp	3.78	3.85	3.79
Dart	2.28	2.33	2.28
Emacs Lisp	8.29	8.41	8.30
Erlang	3.57	3.64	3.58
Fortran	3.93	4.01	3.95
Go	1.99	2.03	2.00
Groovy	2.95	3.01	2.96
Haskell	4.28	4.37	4.28
Java	2.31	2.35	2.31
Pascal	3.67	3.72	3.67
Python	3.22	3.27	3.22
Mean	3.49	3.55	3.50

Context 16K: YaRN				Context 16K: Naive			
Eval/Method	SPLICE _{BM25}	BASELINE	SPLICE _{REPO}	Eval/Method	SPLICE _{BM25}	BASELINE	SPLICE _{REPO}
ArXiv	5.77	5.79	5.77	ArXiv	6.25	6.33	6.25
C	2.68	2.72	2.68	C	2.88	2.96	2.89
C++	2.81	2.85	2.81	C++	3.04	3.13	3.05
CUDA	2.55	2.61	2.56	CUDA	2.84	2.96	2.85
C#	1.92	1.94	1.92	C#	2.04	2.08	2.04
Common Lisp	3.83	3.92	3.84	Common Lisp	4.40	4.56	4.39
Dart	2.30	2.36	2.30	Dart	2.50	2.60	2.51
Emacs Lisp	8.37	8.52	8.38	Emacs Lisp	9.25	9.46	9.25
Erlang	3.60	3.67	3.61	Erlang	3.98	4.10	4.00
Fortran	3.97	4.06	3.99	Fortran	4.56	4.79	4.59
Go	2.00	2.04	2.01	Go	2.14	2.21	2.16
Groovy	2.98	3.03	2.98	Groovy	3.27	3.39	3.28
Haskell	4.32	4.42	4.32	Haskell	4.84	5.03	4.87
Java	2.33	2.37	2.33	Java	2.52	2.60	2.53
Pascal	3.69	3.75	3.69	Pascal	4.05	4.20	4.10
Python	3.24	3.29	3.24	Python	3.55	3.66	3.56
Mean	3.52	3.58	3.53	Mean	3.88	4.00	3.89

Table 14: Perplexity results comparing different ways of organizing the same data. All runs started from the same 270M model with 2048 context and were trained for 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and C code is organized in one of three ways. For details about training please refer to Section 3.1.

Eval/Method	Altered Train Data: C		
	SPLICE _{BM25}	BASELINE	RANDOM
ArXiv	5.46	5.55	5.55
C	2.13	2.17	2.18
C++	2.40	2.47	2.47
CUDA	2.22	2.33	2.33
C#	1.94	2.02	2.02
Common Lisp	2.99	3.20	3.18
Dart	2.14	2.27	2.27
Emacs Lisp	7.86	8.10	8.09
Erlang	2.67	2.77	2.77
Fortran	3.31	3.55	3.56
Go	1.91	2.00	2.00
Groovy	3.01	3.15	3.15
Haskell	3.20	3.37	3.37
Java	2.07	2.16	2.16
Pascal	3.49	3.62	3.64
Python	2.81	2.93	2.93
Mean	3.10	3.23	3.23

C. Ablations

C.1. SPLICE Parameters

There are two important design choices related to SPLICE. First, how many related documents are retrieved in each step (the parameter k in Algorithm 1). Second, how the documents are ordered. Table 15 indicates that $k = 1$ is the best choice, though the differences are rather small. We found that changing the order of documents in training examples hardly matters for 270M models. We use ‘standard’, as ordered by Algorithm 1, the reversed order, and random shuffling.

Table 15: Ablation of SPLICE hyper-parameters. For each ablation, we have trained the same 270M parameter model using different data organization methods. Top- k corresponds to the number of descendants chosen in the $\text{RETRIEVE}(d, k)$ step of the Algorithm 1. Reverse and shuffle correspond to the final order of examples C returned by the Algorithm 1 (reverse – the order of documents in C is reversed, shuffle – examples are shuffled).

Method	Top- k	Order	Code				
			C++	Haskell	Python	CUDA	All
SPLICE _{BM25}	top 1	standard	2.38	3.20	2.82	2.23	2.93
		reverse	2.38	3.21	2.82	2.23	2.93
	top 2	standard	2.39	3.23	2.84	2.24	2.95
		reverse	2.39	3.24	2.84	2.24	2.95
	top 3	standard	2.39	3.24	2.84	2.25	2.97
		reverse	2.39	3.25	2.84	2.25	2.96
		shuffle	2.40	3.24	2.84	2.25	2.96

We also evaluated the influence of BOS and EOS tokens on the performance of trained models. As both REPO and SPLICE methods concatenate documents to create training examples, they effectively slightly decrease the number of separating tokens compared to the BASELINE. However, in Appendix C.2 we included experiments showing that this has no impact on

model performance.

C.2. Importance of Separating Tokens

We also evaluate the influence of BOS and EOS tokens on the performance. To be more precise, in all setups, training examples are separated by BOS and EOS tokens. As SPLICE methods concatenate documents to create training examples, they effectively increase the average example length and decrease the number of separating tokens. To check whether those methods do not simply benefit from the reduction in the number of BOS and EOS tokens, we have trained a model on data prepared similarly as in SPLICE, but instead of most matching documents $\text{RETRIEVE}(d, k)$ returned random documents from the dataset (sampling without replacement). The results are shown in Table 16. We note that the difference between the BASELINE and the random concatenation approach is small, and the random concatenation approach does not result in significant perplexity gains.

Table 16: Perplexity evaluation of two methods of organizing the data. BASELINE – document equals training example. RANDOM – concatenate documents into examples of length bounded by 120K characters. Training examples are then fed into the model and separated by BOS and EOS tokens. The difference is negligible, which suggests that the extension of example length in RANDOM does not help the model to utilize extended context. Experiments were performed using the 270M parameter model. We performed three runs on different subsets of C to provide mean and standard deviation.

Long Context Data	Method	arXiv	Code		Code & arXiv
			Python	All	
C	RANDOM	5.554 ± 0.004	2.931 ± 0.003	3.076 ± 0.005	3.231 ± 0.005
	BASELINE	5.550 ± 0.002	2.931 ± 0.008	3.073 ± 0.006	3.228 ± 0.005

D. Key-Value Retrieval Task

Figure 2 shows how training on SPLICE organized data improves the performance on the key-value retrieval task proposed in (Liu et al., 2023). This is a zero-shot task in which a model is prompted with a JSON dictionary and asked to retrieve a value corresponding to a specified key. The structure of the input is showcased below.

Extract the value corresponding to the specified key in the JSON object below.

```
JSON data:
{"4ef217b7-6bc0-48c6-af35-2765f1e730f3": "068192b7-16b1-40e0-8495-61c63f979d50",
 "cd6b8bdc-bc6c-4490-acb4-bc187a2dccba": "7364a26e-289f-4968-93d3-b273e882bdee",
 "7d057372-4ab8-4811-8110-658c3f19fff4": "3ad075c5-b567-4201-85a7-cb31a0c91540",
 "c62e192d-45e6-4646-bb88-1529c73256c9": "f0411644-1f6d-42a6-8af8-f06da66efc77",
 "06134e93-e158-490e-a66c-8e3b98e12735": "50a26a36-d832-450c-8d6e-a4cc3d0ec0ab",
 "3286f978-4270-4b54-8bfa-540d7e0772e6": "075cc716-1836-4f90-9be3-53e3d4ec6585",
 "4701aa05-c523-4b89-9700-64ab9c37c537": "49d86354-74c4-4256-9b3a-35e6e2b80d00",
 "c8895805-e574-4f13-9fe5-89da1d8c4748": "cc91af7f-8509-4bdc-bad7-2646af68e6d2"}
"4701aa05-c523-4b89-9700-64ab9c37c537":
```

We noted that FoT-trained models struggle with this task. This is probably due to the fact that they extend context only in a couple of layers, and the key-value retrieval task requires looking up and extracting a long sequence of letters and digits. Because of that, we evaluate FoT models with shorter dictionaries consisting of 75 key-value pairs (around 6K tokens) and show the results in Figures 3a, and 3b. For comparison, we also evaluate the 7B CL model with this context length and show the results in Figure 3c.

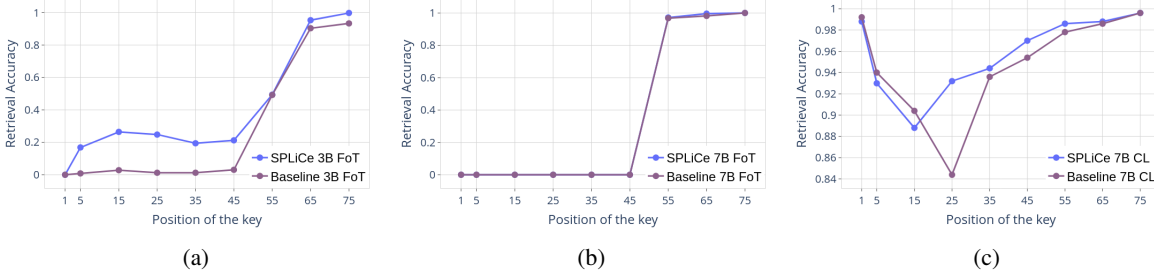


Figure 3: Performance on a smaller version of key-value retrieval task from (Liu et al., 2023). We note that FoT models (a), (b) generally struggle to retrieve tokens that are only visible to a subset of layers with extended context. For comparison, we show the results with a model that has extended context in all layers (c) using CodeLlama (Rozière et al., 2023) method of context extension. Each position was evaluated using 500 examples.

E. Perplexity Improvements

In Figure 4 we present perplexity improvements of 3B FoT SPLiCe_{BM25} over BASELINE. Figure 5 shows the evolution of SPLiCe model perplexity during the training. We follow (Anthropic, 2023) and bucket perplexity by token positions in buckets of length 2^i up to 32768, and then average within the buckets. We average perplexity across arXiv, CUDA, Haskell, and CommonCrawl datasets.

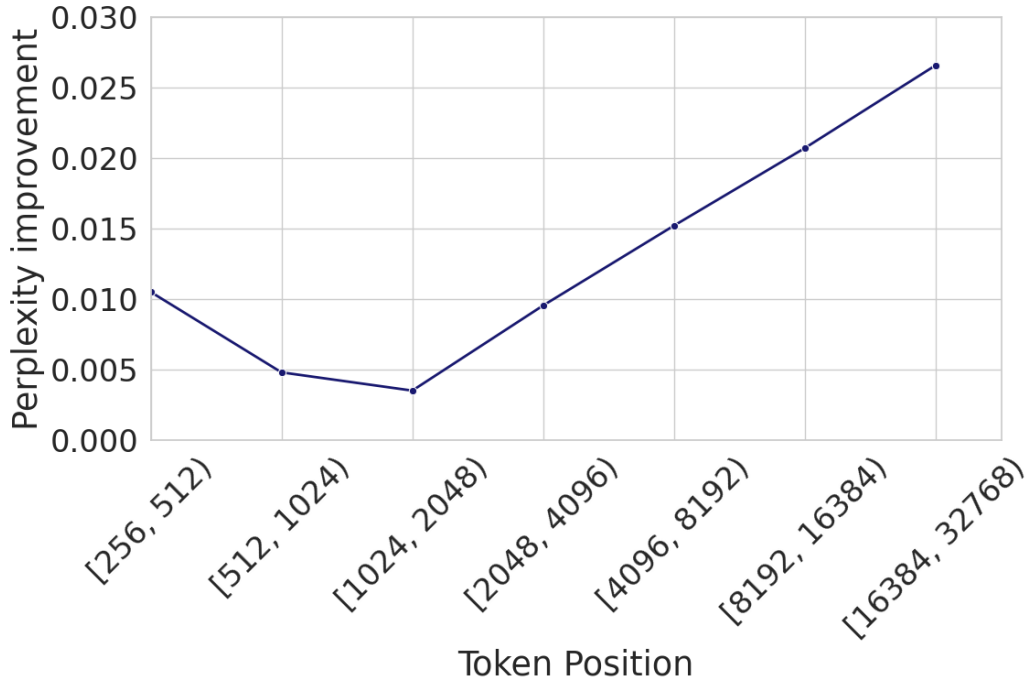


Figure 4: Perplexity improvement with SPLiCe against the BASELINE of the final models (after 21k training steps). We bucket tokens by their positions in the document and calculate the average. Each dot is the difference of the averages of the SPLiCe and BASELINE models. We observe that SPLiCe has smaller perplexity, and the improvements tend to be larger for tokens further in the document.



Figure 5: Evolution of the perplexity with SPLICE, as the model is trained on more tokens. See 4 for the difference with the baseline. As expected, SPLICE significantly improves perplexity for tokens whose positions are very distant in the sequence. Perplexity for more distant tokens improves more significantly compared to tokens in the beginning, early in the training.

F. Shuffling

Table 17: Average classification performance on TREC (Li & Roth, 2002; Hovy et al., 2001). We compare 7B CL model trained on SPLICE prepared data with different approaches to ordering the examples (different function `Order` in 1). SPLICE-shuf denotes the model trained on data that shuffled the documents randomly in context, for SPLICE-no-shuf `Order` was an identity function. We use $\pm$ to denote the standard deviation. We decided to stick with the model trained with random shuffling as it has slightly better long-context performance and lower standard deviation.

TREC			
Model	Context length (# of examples)	SPLICE-no-shuf	SPLICE-shuf
7B CL	32K (1560)	76.0 $\pm$ 2.5	76.6 $\pm$ 1.9
	8K (380)	77.1 $\pm$ 2.4	76.5 $\pm$ 1.8

G. Data Preparation

G.1. Evaluation Data

We have taken a random subset of arXiv from Proof-pile. For StarCoder data, we have downloaded up to 64GB of each of the mentioned language subsets and performed a random 85/15 split for languages that we train on.

When evaluating the perplexity of the model, we skip documents that are shorter than the model context and truncate documents that are longer than that. Table 18 shows the number of tokens over which the perplexity was calculated.

Table 18: Number of evaluation tokens in each of the considered datasets. For each context length c , we consider only documents having not less than c tokens and extract the c tokens prefix.

Eval/Method	16K	32K	64K
ArXiv	16M	16M	16M
C	16M	16M	16M
C++	16M	16M	16M
CUDA	16M	14M	8M
C#	16M	16M	16M
Common Lisp	16M	16M	16M
Dart	16M	16M	16M
Emacs Lisp	16M	15M	9M
Erlang	16M	16M	12M
Fortran	16M	16M	16M
Go	16M	16M	16M
Groovy	11M	4M	2M
Haskell	16M	16M	15M
Java	16M	16M	16M
Pascal	16M	16M	16M
Python	16M	16M	16M

G.2. Train Data

The StackExchange data was taken from the Proof-pile. To prepare the code train data, we take the StarCoder train splits mentioned in Section G.1, shuffle them, group the documents by the repository (documents from the same repository occur one after another), and split them into smaller packs. We also split repos larger than 25MB and filter out files that are longer than 30k characters. The reason behind repo splitting is to avoid the situation where one repository occupies a significant portion of the data pack. We have noticed repos containing as many as 40K files and files as long as 11M characters. The character filtering is consistent with our method as we aim to improve the performance in a scenario that lacks high-quality long-context data. For C# and Python, only one pack is used to organize the data. For C, we have performed a run on three packs and provided results and standard deviation in Table 2. For large models, we run the methods on several packs and concatenate the results into a single dataset. For natural language datasets, we extract a random subset of documents.

H. Faiss Parameters

Our experiments with `SPLICECONT` utilize Faiss (Johnson et al., 2019) for fast approximate inner-product search. To be more precise, we use the "IVF8192,Flat" index that we train on 262144 examples coming from the dataset.

I. Detailed Accuracy Improvements

The performance of in-context learning depends much on the choice of the in-context examples. To study this in more detail we study the following random variable

$$\Delta(c) = \text{ACC}_{\text{SPLICE}}(c) - \text{ACC}_{\text{BASELINE}}(c),$$

where $\text{ACC}_{\text{SPLICE}}(c)$, $\text{ACC}_{\text{BASELINE}}(c)$ are the accuracies of the model trained with `SPLICE` and `BASELINE` respectively on the random choice of in-context examples c . Below we report the histograms of $\delta(c)$. In Table 4 and Table 5 we report mean Δ and 95% confidence intervals as Δ [confidence interval].

Figure 6 shows details about accuracy improvements on TREC when considering different numbers of in-context examples.

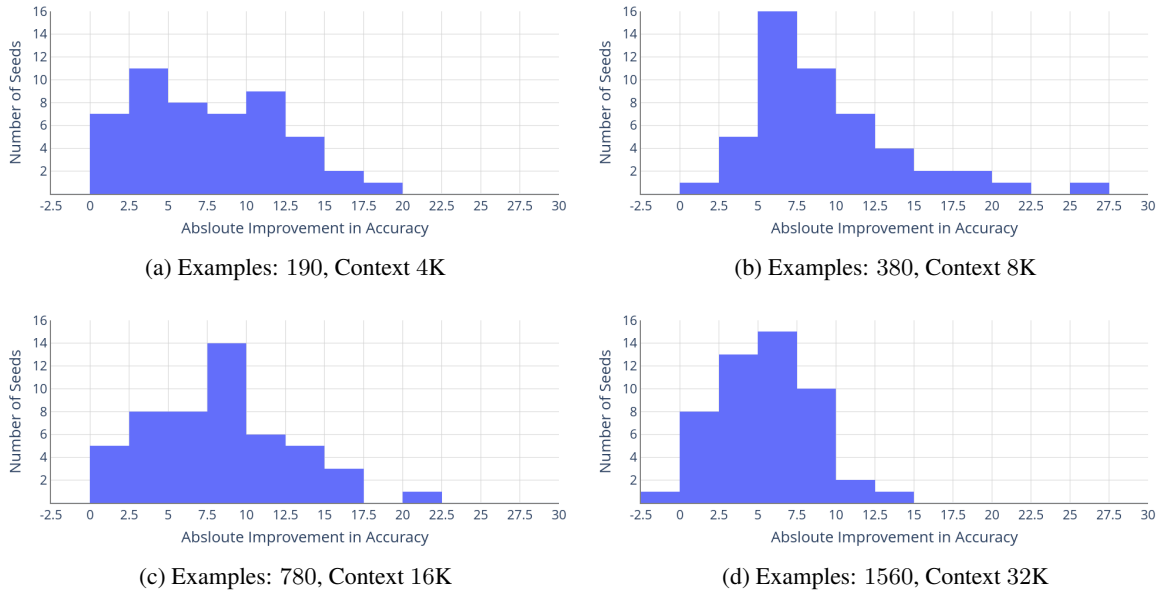


Figure 6: Histograms of accuracy improvement of `SPLICEBM25` over `BASELINE` on TREC question classification task. The results are obtained by comparing the accuracy on the test set of TREC of the 3B FoT model trained with `SPLICE` to the model trained with default data preparation method (`BASELINE`) across 50 sets of in-context examples. Each set of in-context examples consists of elements randomly sampled (without replacement) from the training subset of TREC. Note that the model trained with `SPLICE` is almost always better than the `BASELINE`.

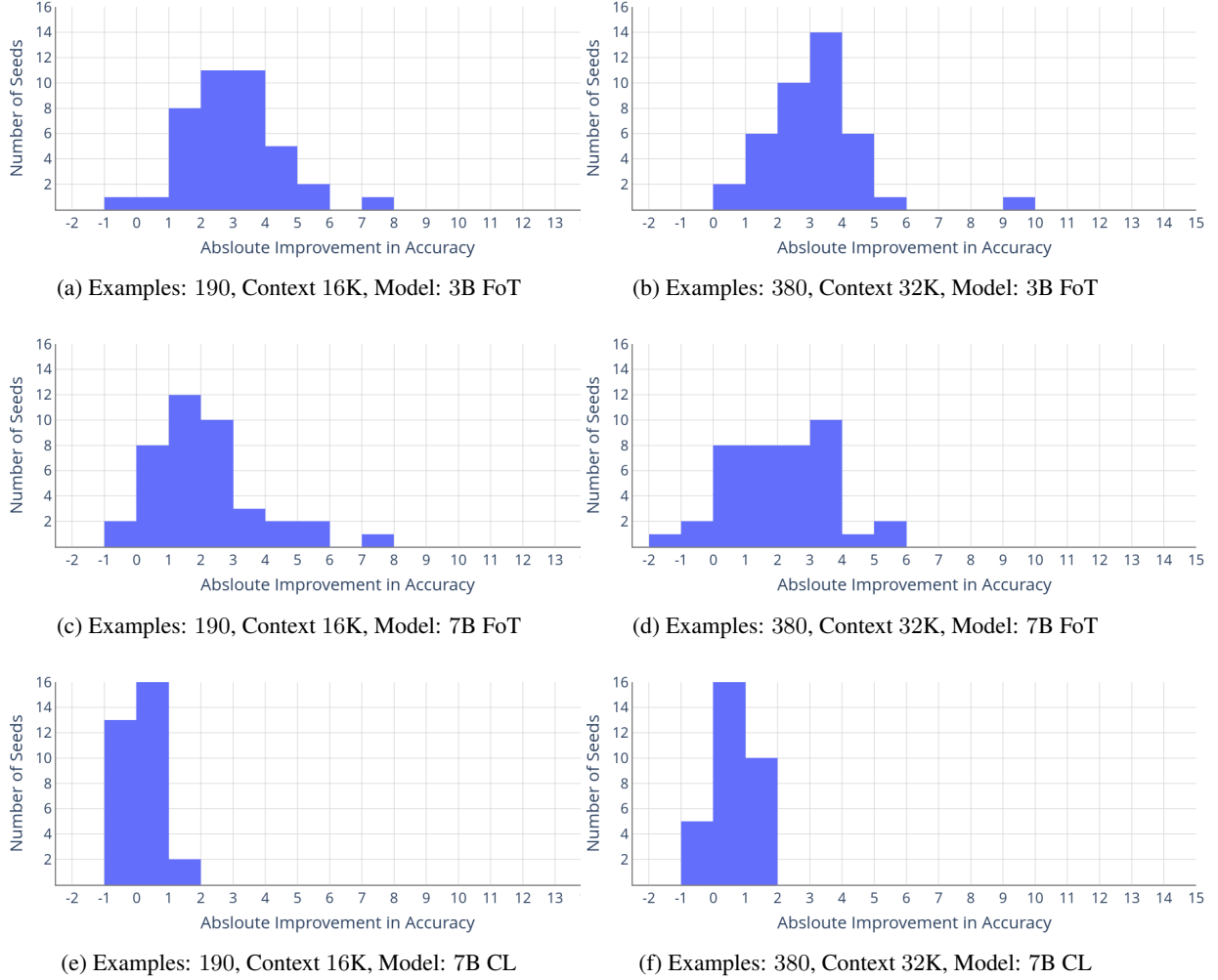


Figure 7: Histograms of accuracy improvement of SPLiCE_{BM25} over BASELINE on DBpedia. We sample 40 sets of in-context examples and, for each set of in-context examples, evaluate on a random 500 element subset of the DBpedia test set.

J. Longer Context

In Table 19, we present the perplexity results for 270M parameter models trained as described in Section 3 but with 2x larger context extension – context extended from 2K to 64K, instead of from 2K to 32K.

Table 19: Perplexity (improvement over BASELINE) for training on a 50/50 data mixture of RedPajama and C# code with longer 64K context.

Long Context Data	Method	arXiv		Code			Code & arXiv
			Haskell	Python	CUDA	All	
C#	SPLiCE _{BM25}	4.86 (.15)	2.60 (.19)	2.66 (.16)	2.32 (.19)	2.75 (.19)	2.88 (.19)
	SPLiCE _{CONT}	4.88 (.13)	2.62 (.17)	2.67 (.15)	2.34 (.17)	2.77 (.17)	2.90 (.17)
	SPLiCE _{REPO}	4.88 (.13)	2.62 (.17)	2.68 (.14)	2.35 (.16)	2.77 (.17)	2.90 (.17)
	BASELINE	5.01	2.79	2.82	2.51	2.94	3.07

K. Data Distribution Property

Chan et al. (2022); Han et al. (2023) shows that the "burstiness" property of training data benefits a model's generalisation. Specifically, "burstiness" presents a flatter frequency distribution, with a relatively higher mass on the rare, long-tail tokens appearing in a sequence, which results in a lower Zipf's coefficient of token frequency. We also observe a burstiness property of data in SPLICE, as shown in Table 20.

Table 20: Zipf's coefficient of token frequency on BASELINE and SPLICE. A lower Zipf's coefficient represents a more significant burstiness property.

Training Data	Context Length	Method	Zipf's Coefficient
C	2K	SPLICE _{BM25}	1.667 _(0.139)
		BASELINE	1.717 _(0.152)
StackEx	2K	SPLICE _{BM25}	1.836 _(0.088)
		BASELINE	1.865 _(0.074)
C	32K	SPLICE _{BM25}	1.512 _(0.055)
		BASELINE	1.593 _(0.025)
StackEx	32K	SPLICE _{BM25}	1.643 _(0.026)
		BASELINE	1.664 _(0.013)