# Two Are Better than One: Context Window Extension with Multi-Grained Self-Injection

Anonymous ACL submission

## Abstract

The limited context window of contemporary large language models (LLMs) hinders broader application. In this work, we present SharedLLM, a novel approach grounded in the design philosophy of multi-grained context compression and query-aware information retrieval. SharedLLM is composed of two short-context LLMs: a lower moel (compressor) and an upper model (decoder). The lower model compresses context information, while the upper model processes compressed, context information from the lower model and performs context-aware modeling. Information transfer between the compressor and decoder occurs only at the lowest layers to reduce redundant computation. Based on this architecture, we introduce a specialized tree-style data structure to efficiently encode, store and retrieve multi-grained contextual information from text chunks. This entire process, wherein the sender and receiver are derived from the same LLM layer, is referred to as *self-injection*. In our evaluation on long-context modeling and understanding tasks, SharedLLM achieves superior or comparable results to several strong baselines, striking an effective balance between efficiency and performance. Meanwhile, with the aforementioned design choices, SharedLLM can greatly reduce memory consumption, and demonstrates substantial speed-ups over other advanced baselines ($2\times$ over streaming, $3\times$ over encoder-decoder architectures). The core code of our implementation along with training and evaluation is available in appendix and supplementary.

## 1 Introduction

Since the release of GPT-3 (Brown, 2020), the rapid advancement of large language models (LLMs) (Chowdhery et al., 2022; Achiam et al., 2023; Touvron et al., 2023a,b; Dubey et al., 2024) has revolutionized the NLP research community and transformed various workflows. Pretrained on trillions of tokens, these models exhibit exceptional capabilities, such as completing unfinished text or code and following human instructions to perform designated tasks after minimal supervised fine-tuning (Wei et al., 2021; Chung et al., 2024).

Despite their impressive abilities, several factors limit their broader application. One major constraint is the *context window* size (Hsieh et al., 2024), which refers to the maximum number of tokens an LLM can process at once. The context window length is implicitly determined during pretraining; for instance, LLaMA and LLaMA-2 have context windows of 2,048 and 4,096 tokens, respectively. When input text exceeds this limit, LLMs may exhibit erratic behavior during inference. Unfortunately, due to GPU memory constraints, high training costs, and the scarcity of long-context training data, LLMs are often pretrained with relatively short context windows. This limitation severely restricts their applicability in tasks requiring longer context windows, such as long-document summarization and information retrieval, where much longer windows are necessary.

Many researchers strive to extend the context window of LLMs while minimizing the time, memory, and training costs during both training and inference. One common approach involves post-pretraining LLMs on long-context corpora using hundreds of GPUs (TogetherAI, 2023; Xiong et al., 2024). Another line of work explores position interpolation (Chen et al., 2023; Peng et al., 2023), which rescales the RoPE (Rotary Position Embedding) frequency and attention scores. While this method is widely used, it still requires long-context continual pretraining. For example, YaRN (Peng et al., 2023) extends LLaMA's context length to 128K tokens by continuing pretraining on 64K-token sequences using full attention. The combination of parameter-efficient fine-tuning (PEFT) and sparse attention (Chen et al., 2024) accelerates tun-

ing but faces challenges with extrapolation. Other approaches like streaming-style architectures (Xiao et al., 2024b; Zhang et al., 2024a; Yen et al., 2024), maintain a constant-sized memory that operates as a sliding window. While this design significantly reduces memory usage, its specialized attention pattern causes incompatibility with high-performance attention implementations like FlashAttention (Dao et al., 2022; Dao, 2023), potentially leading to slower inference speeds. Context compression techniques are also widely explored (Zhang et al., 2024a; Yen et al., 2024). Although they offer high parallelism, improving speed, they tend to consume significant memory, greatly limiting their real applications.

## 2 Related Work

**Long-context Language Models.** There are two prevalent routines to build LLMs that are capable of processing extremely long text: directly pretraining on large corpus of targeted context length from scratch (Touvron et al., 2023a; Dubey et al., 2024; Jiang et al., 2023; GLM et al., 2024) or adapting short context-window LLMs to longer context lengths via combined various techniques (Tworkowski et al., 2024). The former approach consumes tremendous data and computational resources, while the latter allows for more convenience and flexibility for researchers and developers to explore potential optimization to the default settings (Fu et al., 2024). The core idea behind these adaptations is to *mimic* short input scenarios (*i.e.*, length within the model's text window) when the input length exceeds window size. Attention map manipulation is the most common approach for this goal, which can be realized via positional encoding (PE) rescaling, such as ALiBi (Press et al., 2021), positional interpolation (PI) (Chen et al., 2023) and YaRN (Peng et al., 2023), or positional index rearranging (Xiao et al., 2024b; Ding et al., 2023; An et al., 2024; He et al., 2024). Both directly or indirectly adjust attention scores to be similar as the short-input scenarios so that the model can handily deal with. Another line of works compress past tokens sequentially into dense representations (Chevalier et al., 2023; Zhang et al., 2024a) as input at the next step or store them in an *external* retrievable memory (Wu et al., 2022; Xiao et al., 2024a) to reduce the input lengths. (Yen et al., 2024) utilizes small model such as RoBERTa (Liu, 2019) for context encoding to boost speed and enable higher parallelism. However, this heterogeneous architecture necessitates meticulous task design for the extra pretraining and warmup stages to stabilize the fine-tuning process. In contrast to these works, our method directly tunes *off-the-shelf* models to compress context into structural representations for query-aware retrieval. Powered by efficient architecture design and a fast-forwarding mechanism, the whole procedure can be fully paralleled online without excessive memory usage, which greatly cuts down the latency during inference time.

**Efficient Methods for Long-context Modeling.** In vanilla self-attention, the space and time complexity grows quadratically ($O(L^2)$) with the input sequence length $L$, which can cause out-of-memory (*OOM*) issues on GPU clusters. A straightforward solution is to add parameter efficient fine-tuning (PEFT) modules (Chen et al., 2024; Zhang et al., 2024a,b) to shrink the size of gradient tensors during backward propagation. Many works strive to reduce the memory footprint of attention computation to enhance computational efficiency. Longformer (Beltagy et al., 2020) introduces a hybrid attention pattern to capture local and global semantic features concurrently. (Katharopoulos et al., 2020) designs linearized attention that merely demands $O(L)$ space to accomplish attention computation. FlashAttention (Dao et al., 2022; Dao, 2023) and PagedAttention (Kwon et al., 2023) maximize the memory efficiency from system's perspective. More recently, (Xiao et al., 2024b) discovers the "attention sink" phenomenon and proposes streaming-llm to address high perplexity issue in generation under window-attention. Our work basically follows the efficient design principle in three aspects: 1) lightweight architecture through lower layer self-injection; 2) compact structural representations via structural information extraction and compression; 3) efficient construction and retrieval algorithm based on the proposed context tree structure.

## 3 Method

In this section, we first introduce the overall architecture of our proposed SharedLLM in Sec. 3.1, and then elaborate on its two main components, lower model and upper model in Sec. 3.2 and 3.3.

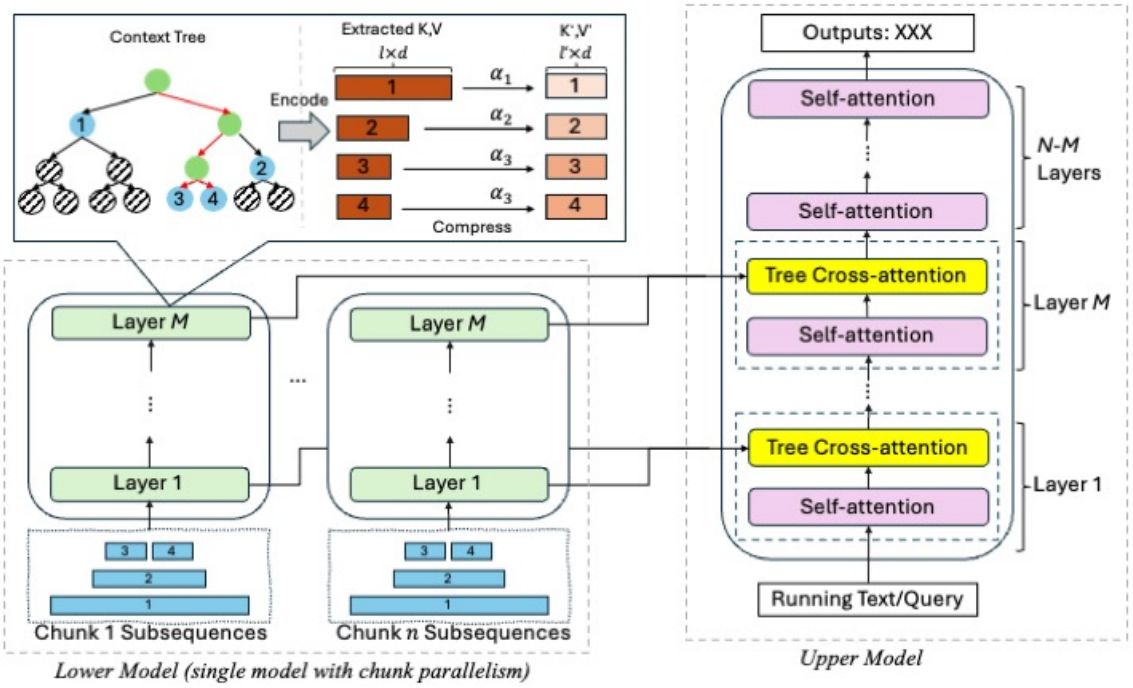


Figure 1: Overview of SharedLLM. It resembles general encoder-decoder architecture like T5 (Raffel et al., 2020). However, the interaction between lower and upper model occurs in the bottom $M$ layers through shared KV states which are encoded and compressed from the context tree nodes (subsequences). The orange arrows mark the paths through which KV states from lower model's self-attention layers are dispatched to the upper model's corresponding cross-attention layers, i.e., the process of *self-injection.*

### 3.1 Overview

As illustrated in Figure 1, SharedLLM adopts a hierarchical architecture, akin but not identical to classical encoder-decoder models. The *lower model*, or the "compressor", breaks down the long input context $X_C$ into smaller chunks that can be processed within limited GPU memory. It then uses the same LLM model to compress each context chunk into compact and structured representations in parallel. The *upper model*, or the "decoder", takes the rear part of the input text (the running context, such as questions) as input, then integrates the compressed information from the lower model, and finally predicts future tokens in an auto-regressive manner.

The lower and upper models are interconnected through shared key-value (KV) states and cross-attention modules between the corresponding layers. To enable efficient and effective information retrieval and integration, the context information processed by the lower model is organized into a binary tree, referred to as the *context tree*, which stores multi-grained information at different levels. This structure allows the upper model to leverage its processed running text to efficiently retrieve relevant information from the binary tree based on a depth-first search algorithm. The retrieved information is then integrated with the input through cross-attention, enabling the model to answer questions or perform language modeling.

In the following, we elaborate on the lower and upper model. To begin with, we first define some notations to enhance clarity and readability. Let $X = \{x_1, x_2, ..., x_T\}$ represent the entire input sequence, where $T$ denotes the sequence length. In accordance with previous setting (Yen et al., 2024), we split these tokens into contiguous parts: $X = \texttt{concat}([X_C; X_D])$, where the past context $X_C$ and the running text $X_D$ serve as inputs to the lower and upper models, respectively. Furthermore, the past context $X_C$ is partioned into $n$ smaller and non-overlapping chunks denoted by $C_1, C_2, ..., C_n$, namely, where $C_1 \cup C_2 \cup ... \cup C_n = X_C$ and $C_i \cap C_j = \emptyset, \forall i \neq j$. The chunk size is controlled to fit within the lower model's context window—e.g., 4,096 tokens for LLaMA-2-7B (Touvron et al., 2023b)—enabling the lower model to fully utilize its encoding capacity.

### 3.2 Lower Model

The lower model is a small pretrained LLM, implemented as the first $M$ shallow layers of LLaMA-

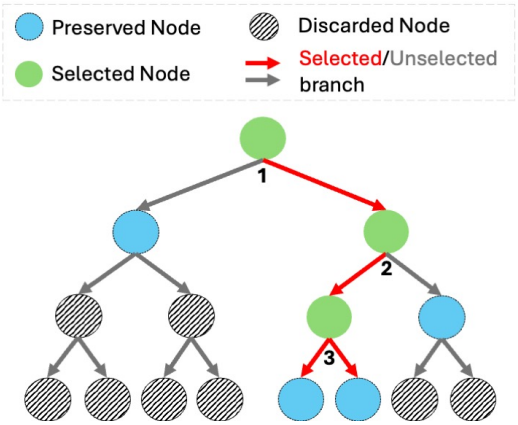


Figure 2: An running example of our tree (depth=3). The digits mark the step indices in the split-and-search procedure.

2. It independently encodes and compresses each past context chunk $C_i$ from the set of chunks $\{C_i\}_{i=1}^n$, and constructs a context tree that stores multi-grained information at different levels. The encoding for all chunks $\{C_i\}_{i=1}^n$ is fully paralleled to boost the speed. Below, we detail the context tree structure and its efficiency-enhanced query-dependent dynamic construction, and the tree search process.

**Context Tree.** The motivation behind building the context tree is both intuitive and problem-driven. Given a text chunk $C_i$ and a task-specific query, task-related information is often distributed unevenly across the chunk. For instance, to summarize a given passage, one should focus more on the topic sentences, extract key messages from them, and rephrase to produce the answer, rather than focusing on narrative details. Whereas in the task of passkey finding, detailed relations are more important than theme paragraphs. To this end, we aim for the contextual representations to capture fine-grained details for the relevant portions of the text, while encoding only coarse-grained information for the less relevant parts. The tree structure is ideal for simulating this process: the splitting of nodes resembles splitting larger text chunks into smaller ones, from which more fine-grained information can be extracted.

In the context tree, its root node contains the entire chunk $C_i = \{x_s, ..., x_t\}$ , where $x_p$ $(s \leq p \leq t)$ denotes a token, $s$ and $t$ are the start and end index of that chunk; and each other node consists of a subsequence of the chunk $C_i$. Then we introduce how to build the child nodes from a parent node. Specifically, for any non-leaf node that contains $l$ tokens $\{x_{u+1}, ..., x_{u+l}\}$, at training phase, we split it into two sub-sequences for constructing its left child and right child as:

$$C_{\texttt{parent}} = \{x_{u+k}\}_{k=1}^{l} \quad (1)$$

$$C_{\texttt{left}} = \{x_{u+k}\}_{k=1}^{b}, C_{\texttt{right}} = \{x_{u+k}\}_{k=b+1}^{l} \quad (2)$$

Here we adopt a random splitting by setting $b = \lfloor \frac{l}{2} - \epsilon \rfloor$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ where $\sigma$ is a predefined hyperparameter, since random lengths can slightly improve the performance as concluded in (Zhang et al., 2024a). At test time, the noise $\epsilon$ is fixed to zero. One can continue this process until arriving at the limited tree depth. Next, building upon this static tree, we construct a more efficient query-dependent dynamic tree.

**Query-Dependent Dynamic Tree Construction and Search.** A task-specific query is typically highly relevant to certain tree nodes, while being less relevant to others. For highly relevant nodes, further expansion is necessary to extract fine-grained information. However, for less relevant nodes, expansion is unnecessary. Thus, instead of building an entire static context tree as mentioned above, we build a query-dependent dynamic tree that expands only the relevant nodes, as shown in Figure 2, significantly saving both GPU memory and time.

Starting from the root node, we perform a depth-first splitting and search process. Each node sequence is first divided into two subsequences according to Eq. (1). We then use a non-parametric policy $\pi$ to decide the next selected node based on the two subsequences, $\boldsymbol{x}_{\texttt{left}}$ and $\boldsymbol{x}_{\texttt{right}}$, and a query sequence $\boldsymbol{y}$:

$$\pi((\boldsymbol{x}_{\texttt{left}}, \boldsymbol{x}_{\texttt{right}}), \boldsymbol{y}) \rightarrow \texttt{left or right}, \quad (3)$$

Here the policy $\pi$ determines whether the left or right child of a node will be selected. The unselected sibling node is marked as “preserved" and will not be expanded further. Note that the root node is always selected to ensure expansion. For policy $\pi$, it is task-specific. Specifically, regarding language modeling task, since there are no explicit queries (i.e., $\boldsymbol{y} = \emptyset$), we simply set $\pi$ to be deterministic:

$$\pi((\boldsymbol{x}_{\texttt{left}}, \boldsymbol{x}_{\texttt{right}}), \boldsymbol{y}) \equiv \texttt{right}. \quad (4)$$

For instruction-following tasks, such as question-answering, where queries like questions are available, $\pi$ selects the node with higher similarity to

the query in the hidden space:

$$\pi((\boldsymbol{x}_{\texttt{left}}, \boldsymbol{x}_{\texttt{right}}), \boldsymbol{y}) = \underset{\phi \in \{\texttt{left},\texttt{right}\}}{\arg\max} (\mathbf{sim}(\boldsymbol{h}_{\boldsymbol{x}_\phi}, \boldsymbol{h}_{\boldsymbol{y}})), \quad (5)$$

where $\mathbf{sim}(\cdot, \cdot)$ represents the cosine similarity between two vectors. The hidden vector $\boldsymbol{h}$ at the last position of a sequence is embedded by the lower or upper model. Specifically, this involves a short forward pass through one self-attention layer in the lower model for $\boldsymbol{h}_{\boldsymbol{x}_\phi}$ and the upper model for $\boldsymbol{h}_{\boldsymbol{y}}$. Once the selected node is determined, the process proceeds with that node, repeating until the leaf nodes are reached. At this point, both the left and right child are marked as "preserved" and will not be expanded further.

For each preserved node, we feed its associated context into the lower model to obtain a collection of key-value (KV) states from all $M$ layers, denoted as $\mathbf{S} = \{\mathbf{K}, \mathbf{V}\}$, where $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{M \times l \times d}$ represent the key and value states for all $M$ layers. Here, $l$ is the sequence length, and $d$ is the hidden dimension. Next, we perform a uniform downsampling along the length dimension to retain only a portion of the KV states, resulting in $\mathbf{S}' = \{\mathbf{K}', \mathbf{V}'\}$, where $\mathbf{K}', \mathbf{V}' \in \mathbb{R}^{M \times l' \times d}$ and $l'$ is the downsampled length. The compression ratio $\alpha$ for the node is defined as $\alpha = l/l'$. For the context tree, we apply a constant compression ratio $\alpha_w$ for all preserved nodes at level $w$, but the ratio decreases progressively from top to bottom, i.e., $\alpha_w > \alpha_{w+1}$. In our implementation, we set $\alpha_w = 2\alpha_{w+1}$. The specific value of $\alpha_w$ can be found in Appendix A.1. This approach creates a *coarse-to-fine* distribution of semantic information from top to down: nodes at higher levels possess longer subsequences and are compressed with a higher compression ratio, corresponding to more coarse-grained information. In contrast, nodes closer to the bottom store finer-grained information.

The overall compression ratio $\beta$ of a tree is defined as the ratio of the chunk length $|C|$ to the total length of the compressed KV states:

$$\beta = \frac{\sum l_w n_w}{\sum l'_w n_w} = \frac{|C|}{\sum l'_w n_w} \quad (6)$$

where $n_w$ is the number of preserved nodes at level $w$, and $l'_w$ is the compressed length of each preserved node at level $w$. For the convenience of parallel processing, we set $\beta$ same for all $n$ context trees. Experimental results in Section 4 demonstrate that this compression ratio can reach as high as 8, significantly improving efficiency.

### 3.3 Upper Model

The upper model primarily inherits from the LLaMA architecture, which consists of $N$ (32 for LLaMA-2-7B) self-attention layers with slight modifications. As illustrated in Figure 1, for each one of the $M$ shallow layers, we add a cross-attention module on top of the standard self-attention layer for information fusion.

**Position-aware Cross-attention on the Context Tree.** In Section 3.2, we obtain a sequence of tree-structured representations $\mathcal{S}' = \{\mathbf{S}'_1, ..., \mathbf{S}'_n\}$ for $n$ chunks $\{C_i\}_{i=1}^n$, where $\mathbf{S}'_i = \{\mathbf{K}'_i, \mathbf{V}'_i\}$ stands for the representations of chunk $C_i$. Since the sequence of chunk keys $\mathcal{K} = \{\mathbf{K}'_1; ...; \mathbf{K}'_n\}$ is produced from ordered chunks $\{C_1, ..., C_n\}$, their positional information should be perceived at chunk level by the query. We assign the following chunk-level positional indices to $\mathbf{Q}$ and $\mathcal{K}$:

$$\mathbf{P}_{\mathbf{Q}} = \{\underbrace{n, ..., n}_{|X_D|}\},$$
$$\mathbf{P}_{\mathcal{K}} = \{\underbrace{0, ..., 0}_{|C_1|/\beta}, \underbrace{1, ..., 1}_{|C_2|/\beta}, \underbrace{n-1, ..., n-1}_{|C_n|/\beta}\} \quad (7)$$

Here we treat the upper model's query $\mathbf{Q}$ as one chunk and assign it the largest positional index, because $\mathbf{Q}$ is encoded from $X_D$, which appears after all the context chunks $X_C$ in the raw input sequence $X$. We will show in Section 4.3 that this setting also facilitates chunk-level extrapolation and answer text production in downstream tasks.

We then conduct cross-attention between the query $\mathbf{Q}$ and the concatenated KVs to integrate their carried context information into the running context for more coherent language modeling:

$$O = \text{cross\_attn}(\mathbf{Q}, \texttt{concat}([\mathbf{K}'_1; ...; \mathbf{K}'_n]), \texttt{concat}([\mathbf{V}'_1; ...; \mathbf{V}'_n])). \quad (8)$$

**Training** We leverage the standard language modeling loss during training, which maximizes the log probability of the ground-truth tokens in the target sequences $X_{\text{tar}}$, conditioned on the context $X_C$ and all preceding tokens $x_{<t}$ from $X_D$:

$$\mathcal{L} = -\sum_{x_t \in X_{\text{tar}}} \log P(x_t | X_C; x_{<t}). \quad (9)$$

For language modeling data, $X_{\text{tar}} = X_D$, i.e., the target tokens are all tokens in $X_D$, excluding the first token. For instruction-following data, $X_D$ includes both the instruction $X_{\text{inst}}$ and the annotated response $X_{\text{res}}$. In this case, we set $X_{\text{tar}} = X_{\text{res}}$, meaning that we optimize only for the response tokens, while the instruction text is masked during loss calculation.

# 4 Experiments

## 4.1 Setup

**Base Model** We select LLaMA-2-7B and LLaMA-3-8B as the base models. Both the upper and lower model are initialized with part of or the whole checkpoint in post-pretraining and with corresponding instruction-tuned version in supervised fine-tuning (SFT), in consistent with previous works (Chen et al., 2024; Yen et al., 2024; Zhang et al., 2024a). We set $M = 4$ in language modeling and $M = 16$ in SFT for lower model.

**Training Dataset** In continual pretraining, we follow (Yen et al., 2024) to prepare the training data by sampling a subset of 20B (1%) tokens from RedPajama's all 7 domains (Together, 2023). Due to the copyright issue, the books3 subset in Books domain (books3 + PG19) is unavailable and thus excluded from our training set, yet we do not renormalize sampling probability across domains. The sampled texts are truncated to 8,192 tokens to form the input. In SFT, we use a mixed dataset, where the input length is filtered to range between 1200 to 8192 tokens, following Zhang et al. (2024a).

**Training Details** We train SharedLLM on an 8× A800 GPU machine. The batch size is set to 1 per GPU with gradient accumulation of 16 steps (global batch size is 128) for continual pretraining and 1 step (global batch size is 8) for SFT. The chunk size is set to 1,024 for language modeling or 512 in SFT, with tree height $h = 3$ and compression ratio $\beta = 8$. For other configurations and hyperparameters, please refer to Appendix A.1 for more details.

**Evaluation Tasks** We assess our method on two typical evaluation tasks in the long-context field: language modeling and long document question–answering. The evaluation on language modeling is performed on the test set of RedPajama, while for long document question–answering we test SharedLLM on three benchmarks. In language modeling task, we measure perplexity of target models on sequence lengths ranging from 4K to 128K using a single A800 80GB GPU. The evaluation covers four datasets: ArXiv, PG19 (Rae et al., 2020), ProofPile (Azerbayev et al., 2024), and CodeParrot (Tunstall et al., 2022) under two settings that utilize different training datasets. Under each setting we test on three out of the four datasets, respectively. The results are posted on Table 1 and 2. All perplexity values in these tables are averaged over 100 examples except for the 128K length, on which we test only 10 examples due to the data scarcity (Yen et al., 2024; Zhang et al., 2024a). For the experiments on encoder-decoder and hierarchical models at 4K length, the input is divided by half (2K/2K) and fed separately into their two submodules.

## 4.2 Main Results

**Language Modeling.** SharedLLM exhibits strong extrapolation capability on both base models—it avoids perplexity explosion even tested on 128K-token length through having seen only at most 8K-length sequences before. In Table 1, SharedLLM almost outperforms other baselines trained on mixed dataset 3-10% under both base models. In Table 2, SharedLLM outperforms CEPE in nearly all cases except 128K context length on ProofPile, showcasing the effectiveness of structural self-injection mechanism. Between the two settings, the improvement over Activation-Beacon is more pronounced than over CEPE, because CEPE experiences an extra pretraining stage to adapt the RoBERTa encoder to the RedPajama corpus and a warmup stage to align the hidden space between encoder and decoder. In contrast, SharedLLM can directly be finetuned from publicly available *off-the-shelf* checkpoints, which saves a great amount of training effort.

**Long-context Understanding Benchmarks.** We continue to test the supervised fine-tuned version of SharedLLM on many downstream tasks from InfiniBench (Zhang et al., 2024c) and LongBench (Bai et al., 2023). Both benchmarks consist of a variety of long-context tasks established from raw and synthetic datasets.

On InfiniBench, we are concentrate on the following tasks: **Math.Find** asks a model to retrieve a special value specified in the prompt (e.g., minimum, maximum, medium, etc.), which examines both the precise retrieval and query understanding

| Base Model | Method | PG19 | | | | ProofPile | | | | CodeParrot | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 4K | 16K | 32K | 100K | 4K | 16K | 32K | 100K | 4K | 16K | 32K | 100K |
| LLaMA-2 | StreamingLLM | 9.21 | 9.25 | 9.24 | 9.32 | 3.47 | 3.51 | 3.50 | 3.55 | 2.55 | 2.60 | 2.54 | 2.56 |
| | LongAlpaca-16K | 9.96 | 9.83 | - | OOM | 3.82 | 3.37 | - | OOM | 2.81 | 2.54 | - | OOM |
| | Activation Beacon | 9.21 | 8.34 | 8.27 | 8.50 | 3.47 | 3.34 | 3.32 | 3.31 | 2.55 | 2.43 | 2.41 | 2.62 |
| | SharedLLM | **8.68** | **8.01** | **7.96** | **8.24** | **3.36** | **3.24** | **3.21** | **3.19** | **2.33** | **2.25** | **2.23** | **2.36** |
| LLaMA-3 | StreamingLLM | 8.75 | 8.81 | 8.83 | 8.87 | 3.29 | 3.38 | 3.40 | 3.44 | 2.39 | 2.48 | 2.43 | 2.46 |
| | LongAlpaca-16K | 8.58 | 8.71 | - | OOM | 3.26 | 3.34 | - | OOM | 2.48 | 2.46 | - | OOM |
| | Activation Beacon | 8.50 | 8.62 | 8.47 | 8.51 | 3.18 | 3.05 | 3.07 | **3.10** | 2.35 | 2.27 | 2.29 | 2.43 |
| | SharedLLM | **8.32** | **8.37** | **8.19** | **8.21** | **3.03** | **2.94** | **2.98** | **3.10** | **2.29** | **2.18** | **2.21** | **2.33** |

Table 1: Perplexity when models are trained on mixed dataset. “OOM" means out-of-memory exception raised during inference. Excessively large perplexities ($> 10^2$) are hidden with a dash (“-").

| Base Model | Arxiv | | | | PG19 | | | | ProofPile | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4K | 8K | 32K | 128K | 4K | 8K | 32K | 128K | 4K | 8K | 32K | 128K |
| LLaMA-2 | 2.60 | - | - | OOM | 6.49 | - | - | OOM | 2.28 | - | - | OOM |
| Positional Interpolation | 3.49 | 3.21 | 2.77 | OOM | 6.97 | 6.77 | 6.89 | OOM | 2.77 | 2.64 | 2.51 | OOM |
| YaRN-2-128K | 3.35 | 3.09 | 2.58 | OOM | 6.85 | 6.62 | 6.91 | OOM | 2.82 | 2.56 | 2.47 | OOM |
| CEPE | 3.03 | 3.02 | 2.51 | 2.97 | 6.69 | 6.40 | 6.80 | 6.10 | 2.38 | 2.43 | 2.45 | **2.39** |
| SharedLLM | **2.99** | **2.97** | **2.46** | **2.91** | **6.55** | **6.28** | **6.65** | **5.96** | **2.33** | **2.34** | **2.38** | 2.40 |

Table 2: Perplexity of models after continual-pretraining on downsampled RedPajama. Best results on *context-extended* models are marked in bold. Perplexity higher than $10^2$ are denoted by dash ("-")

| Base Model | Method | Math.Find | En.MC | Ret.Num |
|---|---|---|---|---|
| LLaMA-2 | LM-Infinite | 5.71 | 30.57 | 4.95 |
| | Str-LLM | 6.00 | 32.31 | 5.23 |
| | InfLLM | 11.14 | 31.44 | 80.58 |
| | SharedLLM | **13.58** | **33.65** | **82.79** |
| LLaMA-3 | LM-Infinite | 22.45 | 45.72 | 7.92 |
| | Str-LLM | 21.89 | 39.95 | 8.17 |
| | InfLLM | 23.95 | 43.58 | 98.85 |
| | SharedLLM | **25.57** | **47.92** | **99.27** |

Table 3: Evaluation of different methods on tasks from InfiniBench.

abilities of the model. **En.MC** instructs a model to collect key information from a extremely long passage and choose the correct answer from many candidate options. **Ret.Num** demands locating repeated hidden numbers in a noisy long context. We compare SharedLLM with advanced baselines capable of extremely long inputs, as shown in Table 3. SharedLLM surpasses many strong baselines on both tasks (2.44/1.62 points on Math.Find, 1.34 points or 4.1% on En.MC, 2.21/0.41 points over state-of-the-arts), showing excellent capabilities in tackling extremely long input.

For LongBench, we report the categorical scores over all 14 English tasks in 5 categories, including single-document QA (SD-QA), multi-document QA (MD-QA), summarization (Sum), few-shot task (FS) and code-completion (Code), as shown in Table 4. SharedLLM outperforms or matches other advanced instruction-tuned long-context baselines across all five categories. Note that for SharedLLM we feed the model with the entire document while for other models documents are truncated from the middle to fit their context length as the code suggests—such an action could “improve" the performance (Zhang et al., 2024a), especially on decoder-only models, as relevant information for many tasks is located at the beginning or end of the entire text rather than the middle part.

### 4.3 Ablation Studies

We consider the following ablative settings to verify the rationale of the design considerations in SharedLLM: 1) tree depth; 2) compression ratio; 3) the collection of context information injection layers; 4) other configurations, including the effect from retrieval policy $\pi$ (only for instruction-following task), the noise in node splitting, and the addition of chunk-level positional indices during cross-attention.

The results are displayed in Table 5, from which we find that both tree depth and compression ratio should be set appropriately to achieve near-optimal performance. For example, SharedLLM performs

| Base Model | Method | SD-QA | MD-QA | Sum | FS | Code | Avg |
|---|---|---|---|---|---|---|---|
| LLaMA-2 | Base | 24.90 | 22.60 | 24.70 | 60.00 | 48.10 | 35.20 |
| | LongAlpaca-16K | **28.70** | 28.10 | **27.80** | 63.70 | 56.00 | 39.78 |
| | YaRN-128K | 24.03 | 24.11 | 19.82 | 60.00 | **62.73** | 36.38 |
| | Activation Beacon | 28.27 | 28.44 | 25.15 | 61.00 | 57.75 | 38.86 |
| | SharedLLM | 28.43 | **30.93** | 25.63 | **63.98** | 59.15 | **40.37** |
| LLaMA-3 | Base | 15.12 | 7.95 | 26.13 | 68.75 | 56.04 | 33.28 |
| | LongAlpaca-16K | 21.41 | 12.45 | 27.74 | 70.72 | 60.05 | 36.93 |
| | YaRN-128K | 17.24 | 10.58 | 22.53 | 69.41 | **63.80** | 34.78 |
| | Activation Beacon | 22.08 | 13.75 | **29.06** | 70.67 | 61.14 | 37.78 |
| | SharedLLM | **22.62** | **14.32** | 28.94 | **71.45** | 63.57 | **38.51** |

Table 4: Evaluation of different methods on LongBench. For each base model and each task, best results are highlighted in bold and second best results are underlined.

best when the tree height is 3. If the height is too small, i.e., the tree is *undersplit* and the chunk size is excessively large so that only coarse-grained context information is retained while task-related fine-grained information is not explicit, or too large, i.e., the tree is *oversplit* and the leaves carry fragmented information which can hardly provide valuable clues for task solving, performance degrades accordingly. A similar trend can be viewed on global compression ratio $\beta$. While abandoning downsampling KV ($\beta = 1$) may bring decline in perplexity, its query-aware information retrieval ability deteriorates. In terms of injection layer selection, our implementation, which is refer to as *continuous bottom*, injects the context information in the bottom $M$ layers. In contrary, *Continuous top* injects context information at the topmost $M$ layers (from layer $N-M+1$ to layer $N$). Interleaving applies cross-attention at regular intervals, such as layer 4, 8, 12, 16... Among these configurations, SharedLLM wins over *continuous top* and *interleaving* on both tasks, indicating the correctness of injection layer selection in SharedLLM.

For other settings, as shown in the bottom rows, removing either of them causes performance drop compared to the default setting, which reveals the contributions of the three design considerations to model's performance. Among these items, the query-aware information retrieval is the core component for the context-tree so that the performance on MD-QA drops mostly after removing it from the network. The sequential order is similarly important and should be perceived during cross-attention to organize the answer accordingly.

Besides the effect on task performance, we also perform more experiments to explore how these configurations impact speed and memory in Appendix C.

| Item | Configuration | Arxiv (32K) | MD-QA |
|---|---|---|---|
| Tree Height | 2 | 2.51 | 30.15 |
| | **3** | **2.46** | **30.93** |
| | 4 | 2.57 | 29.47 |
| Compression Ratio $\beta$ | 1 | **2.43** | 30.55 |
| | 4 | 2.48 | 30.28 |
| | **8** | 2.46 | **30.93** |
| | 16 | 2.52 | 29.81 |
| Injection Layers | **Continuous Bottom** | **2.46** | **30.93** |
| | Continuous Top | 2.61 | 28.66 |
| | Interleaving | 2.57 | 29.15 |
| Other Settings | **Default** | **2.46** | **30.93** |
| | w/o retrieval | - | 29.27 |
| | w/o noise | 2.51 | 30.08 |
| | w/o chunk-level pid | 2.49 | 29.81 |

Table 5: Ablative Studies on different configurations of structural information injection. The best values in each category and settings consistent with our defaults are highlighted in **bold**.

## 5 Conclusion

In this work, we present SharedLLM, which leverages a self-injection mechanism to adapt a pair of short-context LLMs for efficient long-context modeling. By integrating the operations of context compression and key information retrieval into a dedicated binary-tree structure, SharedLLM excels in language modeling and various downstream instruction-following tasks, while maintaining excellent memory and time efficiency. Besides, SharedLLM is directly trained from off-the-shelf LLMs, eliminating the need for additional feature alignment steps and making implementation easier. We hope this learning paradigm can be generalized to other short-context LLMs, offering a scalable approach for a context-window extension to an arbitrary length.

## Limitations

While SharedLLM demonstrates superior performance on both language modeling and long-context benchmarks, as well as high efficiency in terms of time and memory, there are still some limitations. First, although this work strikes a relatively good balance between efficiency and performance at the model architecture level, further improvements could be achieved by optimizing at the system and hardware levels. Second, while a simple and effective retrieval mechanism is implemented in this work, more advanced retrieval techniques, such as BM25 (Robertson et al., 2009) and GraphRAG (Edge et al., 2024), were not explored and may further enhance performance. We aim to pursue these improvements in future research.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. 2024. Training-free long-context scaling of large language models. In *Forty-first International Conference on Machine Learning*.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2024. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. Longlora: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. arxiv 2022. *arXiv preprint arXiv:2204.02311*, 10.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 35th Neural Information Processing Systems Conference (NeurIPS)*.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128k context. *arXiv preprint arXiv:2402.10171*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu,

Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Zhenyu He, Guhao Feng, Shengjie Luo, Kai Yang, Liwei Wang, Jingjing Xu, Zhi Zhang, Hongxia Yang, and Di He. 2024. Two stones hit one bird: Bilevel positional encoding for better length extrapolation. In *Forty-first International Conference on Machine Learning*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Ofir Press, Noah A Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

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

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Together. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

TogetherAI. 2023. Llama-2-7b-32k-instruct - and fine-tuning for llama-2 models with together api.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023a. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. 2022. *Natural language processing with transformers*. " O'Reilly Media, Inc.".

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2024. Focused transformer: Contrastive training for context scaling. *Advances in Neural Information Processing Systems*, 36.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *International Conference on Learning Representations*.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, Song Han, and Maosong Sun. 2024a. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. *arXiv preprint arXiv:2402.04617*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao

Ma. 2024. Effective long-context scaling of foundation models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4643–4663, Mexico City, Mexico. Association for Computational Linguistics.

Howard Yen, Tianyu Gao, and Danqi Chen. 2024. Long-context language modeling with parallel context encoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2588–2610, Bangkok, Thailand. Association for Computational Linguistics.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2024a. Soaring from 4k to 400k: Extending llm's context with activation beacon. *arXiv preprint arXiv:2401.03462*.

Peitian Zhang, Ninglu Shao, Zheng Liu, Shitao Xiao, Hongjin Qian, Qiwei Ye, and Zhicheng Dou. 2024b. Extending llama-3's context ten-fold overnight. *Preprint*, arXiv:2404.19553.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024c. $\infty$Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

# A More Implementation Details

## A.1 Training Configurations

Zero Redundancy Optimizer (ZeRO) stage 3 from DeepSpeed without offload is enabled in both training to distribute the memory allocation among GPUs. The cross-attention layers remain fully tunable, while we opt to train upper model's top $N - M$ self-attention layers in language modeling as post-injection aggregation for faster convergence. No parameter efficient fine-tuning (PEFT) techniques, such as LoRA, are applied during the training time, as PEFT seriously slows down model's convergence (Chen et al., 2024). , which actually requires longer tuning time than partial parameter fine-tuning to reach the optimum. We adopt AdamW optimizer with the starting learning rate $1e^{-5}$ and cosine scheduler during training.

We list more training configurations that are not specified in the main text in Table 6. The sequential values of $\alpha$ are level-wise compression ratio, from level 1 to level 3.

Table 6: Configurations for training on both tasks.

| Item | Language Modeling | Supervised Fine-tuning |
|---|---|---|
| training epoch | 1 | 2 |
| warmup ratio | 0.01 | 0.001 |
| $\sigma$ | l/5 | l/10 |
| chunk size | 1024 | 512 |
| $\alpha$ | $1/16, 1/8, 1/4$ | |
| AdamW $(\beta_1, \beta_2)$ | 0.9, 0.999 | |

## A.2 Online Split-and-Search Algorithm

We provide the pseudo code for the online split-and-search algorithm introduced in Section 3.2, from the splitting of root node till collecting all key-value states for all preserved nodes and all $M$ layers. The code snippet in the entire model.py file can be found in the supplementary material.

## A.3 Dataset Statistics

**Downsampled Redpajama.** We follow (Yen et al., 2024) and (Touvron et al., 2023b) to prepare our training set. The proportions of data regarding seven domains in the resulted training set are listed in Table 7.

**Mixed Dataset in SFT.** This dataset is directly picked from (Zhang et al., 2024a), which is a mixture of RedPajama and LongAlpaca (Chen et al., 2024). We follow (Zhang et al., 2024a) to only filter samples whose lengths range from 1K to 8K. The distribution of samples in terms of lengths is below.

## A.4 Details of Test Benchmarks

**RedPajama** In the test of long-context modeling capability, we use a tiny proportion of corpus which has never been seen by the model during the training time as the test set. The test set is sampled at the same time of sampling for the training set. The sampled passages are ensured to match the corresponding test lengths.

**Long Bench** (Bai et al., 2023) is the first bilingual (English and Chinese), multi-task benchmark for long context understanding. It comprises 21 datasets across 6 subcategories, which aims for a more rigorous evaluation of long context understanding. These cateogries encompass *single document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion*. The average length of documents is 6,711 words in English and 13,386 characters in Chinese.

**Infinity-Bench** (Zhang et al., 2024c) extends context lengths in previous long-context benchmarks from 10K to more than 100K tokens. The benchmark is composed of both synthetic and realistic tasks spanning diverse domains and bilingual (Chinese and English). Specifically, the task categories in $\infty$BENCH include retrival (Ret.), summarization (sum), question answering (QA), code and math.

# B More Experiments

## B.1 Passkey Retrieval

We further assess the retrieval capability of SharedLLM on passkey retrieval task, as known as needle-in-haystack (NIAH). Following the settings in Yen et al. (2024), we train a new version of SharedLLM that can perform accurate passkey retrieval from the haystacks of surrounded nonsense. We follow the examples in Chen et al. (2024) to set up the single key-value pair test cases. The results averaged on 10 random generated NIAH test samples are shown in Figure 3. It can be observed that SharedLLM enjoys the minimal accuracy decay as length extends compared to other baselines, although it has only seen context within 8K length.

## B.2 Other Base Models

Apart from LLaMA series, we also test our approach on Mistral-7B model. The results are shown

```
# N: number of trees; L: chunk size

# depth: tree depth; chunk_ids: the entire input ids for chunk in shape (N, L)
# gamma: a hyper-parameter to adjust the variance of the gaussian sampling

selected_input_ids = chunk_ids
selected_length = chunk_ids.shape[-1]
all_kvs = []

for i in range(depth):
   # sample lengths of left and right child
   if i < depth - 1:
      half_length = last_length // 2
      sigma = half_length / gamma
      delta = random.randn(1) * sigma
      l_left, l_right = half_length - int(delta), half_length + int(delta)

      # split the node into two children
      left_input_ids, right_input_ids = input_ids[:l_left], input_ids[-l_right:]
      # query_aware is a flag indicating if the selected nodes are determined on query
      if query_aware:
         # short forward (1-layer) to get representation vectors for the query and two nodes
         h_q = upper_model(query, 1)
         h_left, h_right = lower_model(left_input_ids, 1), lower_model(right_input_ids, 1)
         selected = argmax(sim(h_q, h_left), sim(h_q, h_right)
      else:
         selected = 1 # deterministic example, can change to 0 or random selection

      selected_input_ids = [left_input_ids, right_input_ids][selected]
      selected_length = [l_left, l_right][selected]

      preserved_input_ids = [left_input_ids, right_input_ids][1 - selected]
   else:
      preserved_input_ids = cat(last_input_ids.chunk(2, -1), 0)

   cur_level_kvs = lower_model(preserved_input_ids).past_key_values
   cur_level_kvs = downsample(cur_level_kvs)
   all_kvs.append(cur_level_kvs)
```

**Algorithm 1:** Pseudo code of dynamic Construction-and-Search.

Table 7: Dataset composition in our downsampled Redpajama (20B) tokens.

| Domain | Proportion (%) |
|---|---|
| Arxiv | 2.5 |
| Books (w/o S3) | 4.5 |
| C4 | 15.0 |
| CommonCrawl | 67.0 |
| Github | 4.5 |
| StackExchange | 2.0 |
| Wikipedia | 4.5 |

Table 8: Proportion of samples within each length interval.

| Length | <2K | 2∼4K | 4∼6K | 6∼8K |
|---|---|---|---|---|
| Proportion | 47% | 29% | 8% | 16% |

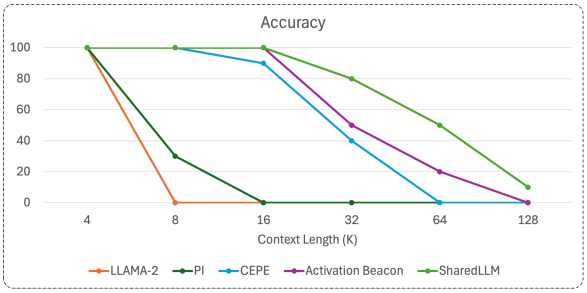


Figure 3: Accuracy comparison on passkey retrieval (single key-value pair) task.

in Table 9. Still, SharedLLM outperforms all baselines on average after changing the base model, which reveals the model generalizability of our proposed method.

## C Time and Memory Efficiency

Apart from strong performance on downstream tasks, SharedLLM demonstrates high computational efficiency in terms of both inference speed and GPU memory utilization. We compare these metrics produced by SharedLLM against other representative models of streaming (Zhang et al., 2024a), encoder-decoder (Yen et al., 2024) and vanilla (Peng et al., 2023) architectures that have shown competitive performance in prior evaluations. The results are visualized in Figure 4.

YaRN (Peng et al., 2023), which exploits the same fully attention as vanilla auto-regressive LLaMA, has $O(L^2)$ time and space complexity. The squared complexity makes it the only model that triggers out-of-memory exception at 128K length. Activation Beacon (Zhang et al., 2024a), which adopts the streaming processing paradigm, maintains a minimum constant memory $O(l)$ under

| Method | SD-QA | MD-QA | Sum | FS | Code | Avg |
|---|---|---|---|---|---|---|
| Base | 23.10 | 16.20 | 23.17 | 48.20 | 46.10 | 30.30 |
| StreamingLLM | 27.19 | 18.15 | 25.37 | 51.85 | 48.98 | 33.26 |
| CEPE | 25.36 | <u>19.03</u> | <u>26.83</u> | <u>52.79</u> | 47.80 | 33.40 |
| Activation Beacon | **29.89** | 18.04 | 25.92 | 52.36 | **52.70** | <u>34.57</u> |
| SharedLLM | <u>29.71</u> | **19.57** | **27.25** | **54.86** | <u>52.39</u> | **35.63** |

Table 9: Results on LongBench with Mistral-7B as the base model. Best results are highlighted in **bold**, while second best results are <u>underlined</u>.

different input lengths $L$, where $l$ is the sliding window length. However, Activation Beacon is incompatible with FlashAttention (Dao, 2023) also due to its specialized attention paradigm, which causes a sharp increment in inference time as input size grows. CEPE can process past context chunks in parallel, but these chunks must be passed through all its encoder layers (24-layer RoBERTa in CEPE) and layer-wise linear projections to obtain the final hidden states for cross-attention, leading to even slower inference speed than non-parallel Activation Beacon. In contrast, SharedLLM avoids such redundancy through shallow-layer compression and injection, which exhibits significant speed-up and limited memory consumption.

We have explained the outstanding efficiency of our model by comparing the memory usage and inference speed with other competitors. In this section, we give a more comprehensive analysis towards the inherent factors that may impact model's efficiency, including compression ratio $\beta$, tree height $h$, the number of shared layers $M$ and the retrieval-based policy which requires an additional short forward pass.

| $M$ | 1 | 2 | **4** | 8 | 16 |
|---|---|---|---|---|---|
| Time (s) | 6.78 | 9.35 | **11.81** | 16.81 | 25.85 |
| Memory (GB) | 21.04 | 21.50 | **22.39** | 24.08 | 27.82 |

Table 10: Inference time under various $M$ with constant $h = 3$ and $\beta = 8$. Our default setting is highlighted in **bold**.

We rerun our experiments to measure the forward time and memory cost from language modeling on 8K tokens, adjusting one variable at a time while keeping others at their default values. The results are shown in Table 10, 11 and 12. Among these factors, the number of injection layers, $M$, has the most significant impact on both speed and memory: both memory and latency grows as $M$ increases. As an opposite, compression ratio $\beta$ and tree height $h$ produces nuances effect on both metrics. For example, if we decreases $\beta$ from 64 to 1 (preserve all KVs), the inference time increases by 6.7% while memory increases by 3%. A similar trend is observed on experiments with tree height $h$. We speculate that the reason behind these outcomes are partly from the internal optimization in FlashAttention, which efficiently computes attention blockwisely. When the configuration meets its requirement for block size and hidden dimension (e.g., length is divisible by 256),

| $\beta$ | 64 | 32 | 16 | **8** | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| Time (s) | 11.68 | 11.73 | 11.78 | **11.81** | 11.87 | 12.04 | 12.47 |
| Memory (GB) | 22.20 | 22.20 | 22.20 | **22.39** | 22.40 | 22.35 | 22.97 |

Table 11: Inference time under various $\beta$ with constant $h = 3$ and $M = 4$. Our default setting is highlighted in **bold**. For $\beta \in \{1, 2\}$, we are not able to set levelwise compression ratios and thus we set the compression ratio same as the $\beta$ for every level of the tree.

Table 12: Inference time under various $h$ with constant $\beta = 8$ and $M = 4$. Our default setting is highlighted in **bold**.

| $h$ | 1 | 2 | **3** | 4 |
|---|---|---|---|---|
| Time (s) | 11.16 | 11.55 | **11.81** | 11.86 |
| Memory (GB) | 19.72 | 22.42 | **22.39** | 22.41 |

We further investigate the potential overhead caused by the extra short forward path query-aware splitting-and-search algorithm. As shown in Table 13, we observe it incurs around 15% overhead in both time and space. We believe this type of overhead can be further eliminated with more careful optimization to the implementation details.

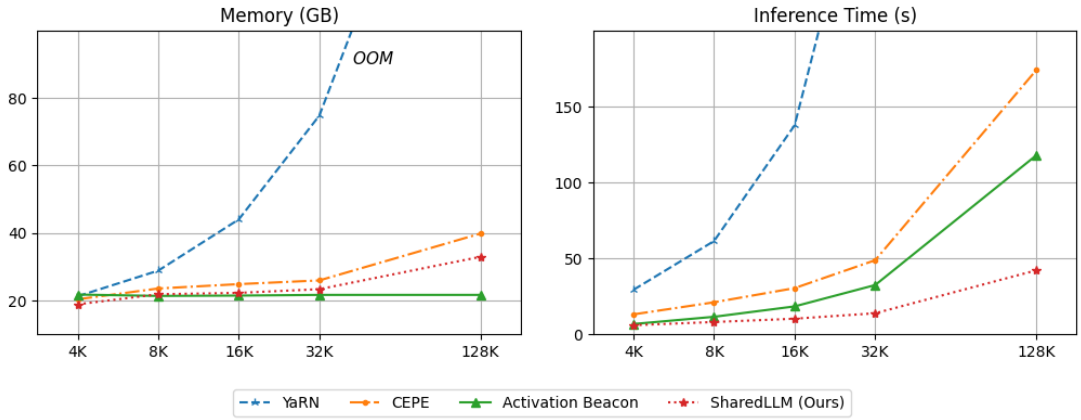


Figure 4: Comparison of memory usage (left) and total inference time on 100 examples (right) between SharedLLM and other recent baselines. The data is collected by running a tiny experiment on 100 examples in corresponding lengths. “OOM" means out-of-memory exception triggered during test time.

Table 13: Comparison of time and memory consumption when query-based retrieval is incorporated/not incorporated in SharedLLM. $h$, $M$ and $\beta$ are fixed at the default values.

| Setting | Time | Memory |
|---|---|---|
| w/o query-aware retrieval | 11.81 | 22.39 |
| w query-aware retrieval | 13.18 | 25.44 |