ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240927

SUBLLM: A Novel Efficient Architecture with Token Sequence Subsampling for LLM

Quandong Wang[a,*,1], Yuxuan Yuan[b,1,2], Xiaoyu Yang[a], Ruike Zhang[c,2], Kang Zhao[a], Wei Liu[a], Jian Luan[a], Daniel Povey[a] and Bin Wang[a]

[a]Xiaomi AI Lab, China
[b]Department of Artificial Intelligence, School of Informatics, Xiamen University, China
[c]Institute of Automation, Chinese Academy of Sciences

Abstract. While Large Language Models (LLMs) have achieved remarkable success in various fields, the efficiency of training and inference remains a major challenge. To address this issue, we propose SUBLLM, short for Subsampling-Upsampling-Bypass Large Language Model, an innovative architecture that extends the core decoder-only framework by incorporating subsampling, upsampling, and bypass modules. The subsampling modules are responsible for shortening the sequence, while the upsampling modules restore the sequence length, and the bypass modules enhance convergence. In comparison to LLaMA, the proposed SUBLLM exhibits significant enhancements in both training and inference speeds as well as memory usage, while maintaining competitive few-shot performance. During training, SUBLLM increases speeds by 26% and cuts memory by 10GB per GPU. In inference, it boosts speeds by up to 37% and reduces memory by 1GB per GPU. The training and inference speeds can be enhanced by 34% and 52% respectively when the context window is expanded to 8192. Our code is available at https://github.com/XiaoMi/subllm.

1 Introduction

Recently in NLP field, the emergence of large language models (LLMs) marks a pivotal advancement in how machines understand and generate human language [5, 26, 37]. Pretrained with huge amounts of parameters on extensive data, LLMs gain extraordinary capabilities across a series of downstream tasks.

Though exhibiting remarkable potential in handling complex tasks, LLMs encounter challenges during training and inference. Firstly, the training process is extremely time-consuming, necessitating the processing of vast amounts of data. Secondly, they need a large amount of GPU memory and computational resources. These factors pose a challenge to their widespread deployment [36, 49].

To address these issues, several approaches have been proposed to accelerate inference and reduce computational costs. Techniques such as distillation [14, 21], quantization [10, 42], and pruning [9, 29]are employed, as well as decoding optimization [19, 32] and conditional computation [2, 18]. Additionally, many efforts are dedicated to training acceleration which often leads to inference acceleration too. Some focus on reducing text redundancy [1, 7, 12, 28] while others tackle the quadratic computational complexity of the Transformer's self-attention mechanism by improving the attention mechanism [40, 45] and proposing new architectures [4, 11, 20, 25].

Drawing from pertinent research [1], natural language tokens in a sequence vary in importance. Selectively identifying and removing less significant tokens can significantly reduce computational demands. Moreover, this targeted approach to prioritizing key information has the potential to enhance training stability, accelerate convergence, and improve overall modeling performance [28].

In this paper, we propose a novel and efficient architecture **S**ubsampling-**U**psampling-**B**ypass **L**arge **L**anguage **M**odel (SUBLLM), inheriting the structure of decoder-only LLM, which dynamically allocates computational resources for tokens according to their importance. SUBLLM integrates subsampling and upsampling modules symmetrically between the Transformer blocks, reducing the computational cost while preserving the input sequence's semantics. Specifically, in the subsampling module, a scoring layer calculates each token's importance as the criterion for token subsampling. Meanwhile, a balancer is adopted to adjust the distribution of the token-level score during training. Subsequently, the upsampling module recovers the subsampled sequences to their prior lengths for token prediction in language modeling. Moreover, to improve training stability and accelerate convergence speed, SUBLLM integrates a bypass module that performs a weighted sum of the upsampled token sequence and the original one. The experimental results compared with LLaMA [37] demonstrate the effectiveness of our proposed SUBLLM on model efficiency as well as performance maintenance. The main contributions of this work are summarized as follows:

- We propose a novel architecture, SUBLLM, which incorporates subsampling, upsampling, and a bypass module. This design dynamically allocates resources to important tokens, reducing computational costs associated with token redundancy and accelerating model convergence through the bypass connection.
- We propose a novel approach to token sequence subsampling that effectively measures token importance scores and controls the distribution of score values as expected, thereby achieving the desired subsampling retention ratio during inference.
- Experimental results demonstrate that SUBLLM achieves 26% and 37% speed-up on training and inference respectively compared to the LLaMA model, with a significant reduction of memory cost, while maintaining the performance.

[*] Corresponding Author. Email: wangquandong@xiaomi.com.
[1] Equal contribution.
[2] Work was done when interning at Xiaomi AI Lab.

2 Related Work

2.1 Training Acceleration

To reduce the computational cost in LLM training, a lot of work has been carried out from the perspective of reducing redundancy in text. Funnel-Transformer [7] uses strided mean pooling to gradually compress the sequence of hidden states in self-attention, and Fourier-Transformer [12] progressively subsamples hidden states with the Fast Fourier Transform operator. Same to these methods, our proposed SUBLLM subsamples the tokens to a shortened sequence. Unlike these methods that reconstruct a sequence by repeating the reduced sequence and adding it back to the original for the final result, SUBLLM's upsampling module takes a different approach. It interpolates between the original and subsampled sequences using token scores as weights, offering a more refined handling of sequence information.

Some other work leverages conditional computation to dynamically expend resources when needed. CoLT5 [1] uses conditional routing to decide whether a given token passes through a light branch or a heavy branch in feedforward and attention layers, so as to devote more resources to important tokens. Further, MoD [28] utilizes a static compute budget, using a per-block router to select tokens for computations, and optimizing FLOP usage by choosing between self-attention and MLP blocks or a residual connection. Our method can also be seen as conditional computation, which dynamically allocates the computational resources to tokens tailored to their importance.

Another type of work focuses on solving the inefficiency problem caused by the attention mechanism when transformers process sequences. Some works focus on improving the attention mechanism to increase the training efficiency [40, 45]. More recent efforts have introduced novel model architectures to overcome this limitation. RWKV [25] addresses limitations in Transformers by replacing quadratic QK attention with a linear scalar formulation. RecurrentGemma [4] combines linear recurrences with local attention to achieve high performance on language tasks with reduced memory usage and faster inference on long sequences. Mamba [11] introduces selective state space models, allowing the model to filter out irrelevant information based on the input, enhancing content-based reasoning. MEGALODON [20] introduces an enhanced MEGA architecture with several novel technical components designed to improve capability, efficiency, and scalability. While MEGALODON accelerates training for long sequence inputs with a 32K window length, its training speed at a 4K window length is slower than that of LLaMA. In contrast, our proposed SUBLLM builds upon the LLaMA model structure by incorporating subsampling modules to reduce sequence length. As a result, SUBLLM surpasses LLaMA in training efficiency, starting from a window length of 2K.

2.2 Inference Acceleration

The parameters that define LLMs are both vast and complex, leading to an ever-increasing demand for computational power and memory capacity. To address these challenges, prior research has primarily concentrated on developing more lightweight models derived from their heavier pre-trained counterparts. Key techniques employed in this endeavor include knowledge distillation [14, 21], quantization [10, 42] and pruning [9, 29]. However, these methods usually seek the balance between effect and inference acceleration at the cost of sacrificing model performance. While the novel model structure SUBLLM we proposed can not only accelerate the inference process but also maintain competitive performance.

Beyond the sheer scale of LLMs, a significant challenge impacting inference speed is the autoregressive decoding process. The language model decodes text sequentially, requiring K serial iterations to generate K tokens. This step-by-step processing not only delays the response time but also turns into a major bottleneck because of the limitations of memory bandwidth. Numerous efforts have been made to optimize the decoding process. For instance, speculative decoding [19, 22, 32] samples from multiple tokens generate from efficient approximation model concurrently as speculative prefixes for the large model. LLMA [43] accelerates the decoding process by identifying and utilizing overlapping text spans between the LLM output and the reference document. Medusa [6] expands the model's predictive capabilities through additional heads and a specific tree-based attention mechanism. MInference [15] accelerates the pre-filling stage by leveraging dynamic sparse attention with spatial aggregation patterns. YOCO [34] optimizes inference efficiency by reusing global KV caches through cross-attention. Reducing the KV cache is also an effective method for accelerating inference [17, 23, 47].

In this work, our proposed new architecture SUBLLM is not mutually exclusive with the inference acceleration method above. On the contrary, SUBLLM can also leverage the previously mentioned strategies to expedite the inference process and reduce memory cost.

3 SUBLLM

As illustrated in Figure 1, the proposed SUBLLM model is based on the decoder-only LLM architecture. To manage the number of tokens processed, subsampling and upsampling modules are integrated into the Transformer blocks. The operation proceeds as follows: Initially, the model uses several Transformer blocks to process the full sequence, capturing a comprehensive token sequence representation. Subsampling modules are then introduced, which sequentially eliminate the less critical tokens, thereby reducing the sequence length required for processing. The highest level of sequence compression occurs in the network's middle blocks. Subsequent to this, upsampling modules are employed to reinstate the sequence length. These modules merge the shorter, processed sequences with the original sequences before subsampling, restoring them to their full lengths. This mechanism allows the decoder-only model to operate as a language model, generating tokens sequentially, which is characteristic of language models where the input and output sequence lengths are identical. Additionally, we have incorporated bypass connection modules after the upsampling process to utilize each pre-subsampling embedding, assisting to improve the learning process from subsampling to upsampling. Our subsequent experiments confirm that this approach significantly improves convergence efficiency.

For better understanding, we here explain the detailed configurations of SUBLLM. As shown in Table 1, let L be the Transformer block, S_i, U_i and B_i are the corresponding subsampling, upsampling module, and bypass module resepectively. The number before L indicates the number of consecutive Transformer blocks. Based on the times of subsampling, we evenly divide the number of blocks in the model following the principles of symmetry and minimizing variance between different parts. Taking a model with 24 blocks as an example, we strategically place subsampling modules after the outputs of the 5th and 10th blocks, and subsequently put upsampling modules and bypass modules after the outputs of the 14th and 19th blocks.

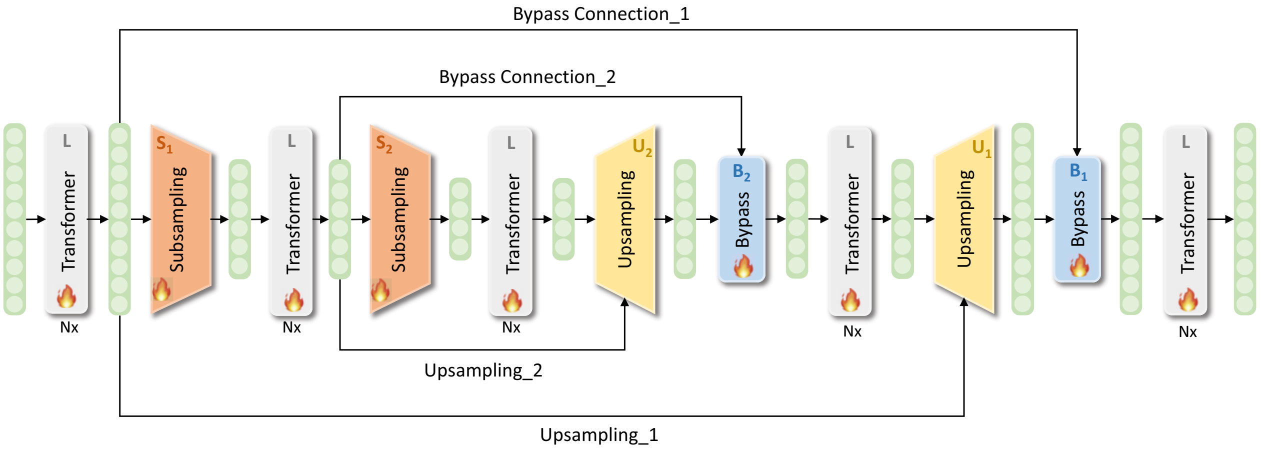


Figure 1. The overall architecture of SUBLLM.

Table 1. The structure of the SUBLLM model is represented with a string. The subsample, upsample and bypass module with the same indexes are paired.

Blocks	S/U Num	Model representation
15	1	$5L\_S_1\_5L\_U_1\_B_1\_5L$
15	2	$3L\_S_1\_3L\_S_2\_3L\_U_2\_B_2\_3L\_U_1\_B_1\_3L$
24	2	$5L\_S_1\_5L\_S_2\_4L\_U_2\_B_2\_5L\_U_1\_B_1\_5L$

3.1 Learnable Subsampling Module

The subsampling module consists of a scoring layer and an activation balancer. Given an input token sequence $\mathbf{x} = \mathbf{x}_1, \dots, \mathbf{x}_N$ of length N, the subsampling module reduces the sequence length by discarding redundant tokens. Let the subsampling retention ratio be d (d smaller than 1), the subsampled sequence $\mathbf{x}'$ is

$$\mathbf{x}' = \textsc{IndexSelect}(\mathbf{x}, \mathcal{I}) \quad (1)$$

$$= x_{\mathcal{I}_1}, \dots, x_{\mathcal{I}_{N'}} \quad (2)$$

where $\mathcal{I}$ is the set of indexes of the kept tokens after subsampling and $N' = \lceil N * d \rceil$ is the length of the subsampled sequence.

Token Selection The indexes of the kept tokens are determined by their importance values. To evaluate the importance of each token, a scoring layer predicts the token-level scalar importance value $\mathbf{w} = w_1, \dots, w_N$, also known as weight:

$$\mathbf{s} = s_1, \dots, s_N \quad (3)$$

$$= \mathcal{S}(\mathbf{x}_1), \dots, \mathcal{S}(\mathbf{x}_N) \quad (4)$$

$$w_n = \textsc{Clamp}(\textsc{Balancer}(s_n, [0, 1])) \quad (5)$$

where $\mathcal{S}$ is the scoring layer, $\mathbf{s}$ is the token-level score, Clamp is the operation clamping the n-th token's score value s_n to $[0, 1]$ and Balancer is a balancer module controlling the distribution of the $\mathbf{s}$. As the same to the decoder-only language model, the scoring layer should not rely on the future tokens. In this work, $\mathcal{S}$ adopts the simplest structure: a single linear layer mapping the dimension of Transformer embedding to a scalar value. During training, the kept indexes $\mathcal{I}$ and discarded indexes $\hat{\mathcal{I}}$ can be formulated as:

$$\mathcal{I} = \{i | w_i \in \textsc{TopK}(\mathbf{w}, N')\} \quad (6)$$

$$\hat{\mathcal{I}} = \{i | w_i \notin \textsc{TopK}(\mathbf{w}, N')\} \quad (7)$$

where $\textsc{TopK}(\mathbf{w}, N')$ stands for the top N' values among $\mathbf{w}$. We keep the original sequential order after subsampling, i.e. the elements in $\mathcal{I}$ are sorted. Note that the TopK operation is cumbersome to implement during inference as previously discarded tokens can be among the top-K tokens after the language model has emitted a few less important tokens. A different token selection strategy is employed in the inference mode to circumvent this problem, which will be discussed in the later section.

Positional Encoding Subsampling We use RoPE [33] for relative positional encoding. After subsampling, tokens that were originally distant might become adjacent, distorting the positional encoding if the subsampled sequence is treated as a new sequence. To address this, we store the indexes of the retained tokens, $\mathcal{I}$, and use them to subsample the sine and cosine matrices used in the RoPE module along the sequence dimension. This ensures that the relative positional information in the subsampled sequence remains consistent with the original sequence. The relative positional encoding of a token pair (i, j) in the subsampled sequence $\mathbf{x}'$ is formulated as follows:

$$RelPos(\mathbf{x}'_i, \mathbf{x}'_j) = \textsc{RoPE}(\mathbf{x}'_i, \mathbf{x}'_j, (\mathcal{I}_i - \mathcal{I}_j)) \quad (8)$$

where RoPE stands for the RoPE module in LLaMA family models.

Inference Mode As mentioned earlier, it is impossible to apply Equation 7 to select the important tokens as the entire token sequence is not available due to the auto-regressive nature of the language model. To tackle this issue, an approximation can be applied to obtain the indexes of the kept tokens:

$$\mathcal{I}_{infer} = \{i | w_i \geq v\} \quad (9)$$

where v is a hyper-parameter between 0 and 1. Tuning v is equivalent to adjusting the balance between the actual inference speed and the model accuracy. The larger v is, the less tokens are kept after subsampling, leading to a faster inference speed. An ideal value of v is supposed to make the actual subsampling retention ratio during inference as close as possible to d.

Balancer Due to the approximation applied in Equation 9, the kept proportion of tokens is changing dynamically and can differ from d. To reduce the gap between training and inference, it is important to keep the proportion of $\mathbf{w} \geq v$ close to d. To encourage this behavior, a balancer [44] module is added before the clamping operation to control the maximum/minimum proportion of positive values of **s**, denoted as p_{max} and p_{min}, and the upper/lower bound of the mean absolute weight values of **s**, denoted as a_{max} and a_{min}. This is achieved by adding an extra gradient term enforcing the distribution of **s** to the gradient back-propagated to **s**. Suppose the subsampling retention ratio is d, the setting of the balancer module is as follows:

$$p_{max} = d + 0.05, p_{min} = d - 0.05 \tag{10}$$

$$a_{max} = 4.0, a_{min} = 1.0 \tag{11}$$

Intuitively, the balancer limits the proportion of positive score values to the range $d \pm 0.05$ during training. In this case, using $v = 0$ as threshold in Equation 9 makes the inference behavior close to the training setting, which is also adopted in out final implementation. Limiting the upper bound of the score value ($a_{max} = 4$) before clamping prevents a too-large and too-sparse gradient during back-propagation. Note that the balancer does not have learnable parameters and is only activated during training, so **s** is untouched during inference.

3.2 Upsampling Module

The upsampling module reconstructs a subsampled token sequence to its original length prior to subsampling. Let S_n be a subsampler subsampling $\mathbf{x}$ of length N to $\mathbf{x}'$ of length N', its paired upsampler U_n utilizes the token indexes $\mathcal{I}$ of $\mathbf{x}'$, the token-level weights $\mathbf{w}$ of $\mathbf{x}$ and token sequence $\mathbf{x}$ to produce a new token sequence $\mathbf{x}_{new}$ of length N following the procedures below. First, a token-level scaling factor $\mathbf{w}_{scaling}$ of length N' is computed:

$$\mathbf{w}_{kept} = \textsc{IndexSelect}(\mathbf{w}, \mathcal{I}) \tag{12}$$

$$\mathbf{w}_{discarded} = \textsc{IndexSelect}(\mathbf{w}, \hat{\mathcal{I}}) \tag{13}$$

$$\mathbf{w}_{sample_i} \sim \textsc{Uniform}(\mathcal{E}), \text{ for } i = 1, 2, ..., N' \tag{14}$$

$$\mathbf{w}_{scaling} = \mathbf{w}_{kept} - \mathbf{w}_{sample} \tag{15}$$

where $\mathcal{E}$ is the set of the elements in $\mathbf{w}_{discarded}$. This enables arbitrary subsampling rate as the length of $\mathbf{w}_{kept}$ and $\mathbf{w}_{discarded}$ are unnecessarily the same. $\mathbf{w}_{scaling}$ serves as a scaling factor when constructing the upsampled sequence $\mathbf{x}_{new}$:

$$\mathbf{x}_{new,i} = \begin{cases} \mathbf{w}_{scaling,i} * \mathcal{G}(\mathbf{x}_i) + (1 - \mathbf{w}_{scaling,i}) * \mathbf{x}_i, & \text{if } i \in \mathcal{I} \\ \mathbf{x}_i, & \text{otherwise} \end{cases} \tag{16}$$

where $\mathcal{G}$ represents the intermediate transformations that $\mathbf{x}_i$ goes through after the subsampler S_n and before the upsampler U_n (e.g. several Transformer blocks or other nested down/upsamplers). Note that instead of directly using $\mathbf{w}_{kept}$, we employed $\mathbf{w}_{scaling}$ to scale $\mathcal{G}(\mathbf{x}_i)$, which is obtained by subtracting the weights of the discarded tokens from the kept tokens. The motivation for this strategy is twofold. First, the subtraction makes the selection of the tokens fully differentiable as the gradient associated with the discarded tokens can be back-propagated. Second, the scoring layer in the subsampler could learn to emit a large score (i.e. 1 after clamping) for all tokens without a penalizing measure. Using $\mathbf{w}_{scaling}$ discourages the model from this behaviour by penalizing the weight values of the discarded tokens, promoting the model to discriminate between more important and less important tokens. The reconstructed sequence $\mathbf{x}_{new}$ can be interpreted an interpolation between the subsampled sequence and the original token sequence before subsampling. As a result, the discarded tokens go through fewer Transformer blocks than the kept tokens. In extreme cases where $\mathbf{w}_i = 1$ for all $i \in \mathcal{I}$, and $\mathbf{w}_i = 0$ for all $i \in \hat{\mathcal{I}}$, the upsampled token sequence is just a index-level re-ordering of $\mathcal{G}(\mathbf{x})$ and $\mathbf{x}$ as the original token order.

3.3 Bypass Module

A bypass module is added to combine the output $\mathbf{y}$ of a group of modules with its input $\mathbf{x}$. It learns a channel-wise weight $\mathbf{c} \in \mathbb{R}^C$ between $[0, 1]$ to control the throughput of each Transformer block:

$$\mathbf{y} = (1 - \mathbf{c}) \odot \mathbf{x} + \mathbf{c} \odot \mathbf{y} \tag{17}$$

where C is the feature dimension of $\mathbf{y}$ and $\odot$ represents the channel-wise multiplication. A larger $\mathbf{c}$ makes the model "straight-through" by increasing the contribution of $\mathbf{y}$. In SUBLLM, one bypass module is added to each paired subsample/upsample modules, i.e. combining the input of S_i with the output of U_i. Bypass module accelerates the convergence of SUBLLM by enforcing all layers to learn high-quality representations, especially at the beginning of the training. A value range can be applied to limit all entries $\mathbf{c}_j$ of $\mathbf{c}$ to the range $[c_{min}, c_{max}]$. This is achieved by negating the positive gradient w.r.t to $\mathbf{c}_j$ if $\mathbf{c}_j$ is smaller than c_{min}, or negating the negative gradient w.r.t to $\mathbf{c}_j$ if $\mathbf{c}_j$ is bigger than c_{max}.

4 Experiments

4.1 Settings

Pre-Training Corpora We use SlimPajama [30] as the pre-training corpus, which includes CommonCrawl, C4, Wikipedia, GitHub, StackExchange, ArXiv, and Book datasets, sampled according to SlimPajama's original proportions.

Pre-Training Details We adopt LLaMA2 [38] as the baseline, training 1.3B [41] and 0.25B parameter versions. SUBLLM shares the same training configuration but introduces only 8,192 additional parameters for the 1.3B model and 4,096 for the 0.25B model. Each model is trained with 100 times the number of its parameters in tokens and has versions with context window sizes of 2K, 4K, and 8K. Eden [44] is used for the learning rate schedule, ScaledAdam [44] as the optimizer, and ZeRO [27] to enhance training efficiency and optimize resource utilization. Training is conducted with bf16 precision using Fairseq [24], with Flash Attention [8] to accelerate the process. More details are in the supplementary material [39].

4.2 Main Results

Table 2 provides the experimental results of LLaMA and the proposed SUBLLM on the computational efficiency during the pre-training and inference phases, as well as model performance. Both models are with the same configuration of 1.3B model size and 4K context window. The minimal retention ratio of the input tokens in SUBLLM subsampling is 40%, which will be discussed in detail in the following section.

Computational Efficiency We explore the computational resource savings of our model, specifically focusing on training and inference acceleration as well as GPU memory reduction. Pre-training speed-up, which is evaluated in the number of tokens that each GPU processes for each second (i.e. TGS), reveals a 26% increase for SUBLLM compared to LLaMA with the same batch size. Meanwhile, the pre-training of SUBLLM achieves a significant memory reduction of 10GB compared with LLaMA. The improvement in pre-training speed is further enhanced when the memory saved by SUBLLM is reallocated to increase the batch size, boosting the training acceleration from 26% to 31%, which we marked as max speed-up.

As for inference acceleration, SUBLLM displays a 37% increase in speed, higher than the 26% improvement observed during training.

Table 2. Experimental results of LLaMA baseline and our proposed SUBLLM on computational efficiency and performances. For the evaluation of computational efficiency, speed-up and memory reduction during both pre-training and inference serve as metrics. TGS represents the number of processing tokens per GPU per second. Inference speed is the number of processed tokens per second. For the evaluation of model performance, we consider valid loss during pre-training and few-shot learning in downstream tasks.

Efficiency	**Pre-Training**					**Inference**		
	Speed-Up			Max Speed-Up				
	TGS↑	Ratio↑	Mem (GB)↓	TGS↑	Ratio↑	Speed↑	Ratio↑	Mem (GB)↓
LLaMA	16,976	×1.00	65.99	18,856	×1.00	17.83	×1.00	18.49
SUBLLM	**21,341**	×**1.26**	**55.81**	**24,773**	×**1.31**	**24.43**	×**1.37**	**17.29**
Performance	**Pre-Training**	**Few-Shot Learning**						
	Valid Loss↓	SST2↑	Amazon↑	DBpedia↑	AGNews↑	Yelp↑	Hate↑	Avg.↑
LLaMA	3.11	81.01	86.54	**45.70**	64.77	87.59	**45.18**	68.47
SUBLLM	**3.10**	**91.95**	**94.57**	42.97	**66.05**	**94.24**	32.23	**70.34**

Table 3. Ablation results of the proposed bypass module of SUBLLM.

Variant	**Valid Loss↓**
SUBLLM	**3.66**
- Bypass Module + Residual Connection	3.72
- Bypass Module	3.73
LLaMA	3.69

Because in trainig stage, SUBLLM only accelerate the forward and backward process rather other computations like parameters update. For clarity, the referenced decoding speed specifically pertains to the decoding of non-first tokens on a single GPU. Also, SUBLLM contributes to 1GB memory reduction compared with LLaMA. These results indicate SUBLLM is a valuable architecture for tasks requiring high-level computational efficiency.

Model Performance In evaluating the model performance of SUBLLM, we consider both the valid loss during pre-training and its performance on few-shot learning tasks. As shown in Table 2, SUBLLM's valid loss is on par with that of LLaMA, indicating that its token prediction capabilities are comparable. For few-shot learning, we evaluate SUBLLM on 6 text classification datasets including sentiment classification (SST2 [31], Amazon and Yelp [46]), topic classification (DBpedia [16], AGNews [46]) and hate speech detection (Hate [3]). Despite some fluctuations in scores across different datasets, the overall performance of SUBLLM is broadly equivalent to LLaMA. Both findings above indicate the validity of the optimized architecture on the model performance in token prediction as well as few-shot in-context learning. The results of the model on other benchmarks [13, 35, 48] can be found in Table 8 in the supplementary materials [39].

4.3 Ablation Study

We perform an ablation study on the 0.25B SUBLLM model to examine the effects of the bypass module on validation loss. The findings summarized in Table 3 show that SUBLLM with all enhancements achieves the lowest valid loss of 3.66. Changing the bypass module's operation from a weighted sum to a standard residual connection increases the validation loss, which is higher than LLaMA. This results demonstrates the importance of weighted integration. Completely removing the bypass module leads to a further increase in validation loss, which confirms the bypass module's role in maintaining low valid loss by using intermediate token information. Overall, the bypass module significantly enhances learning efficiency.

5 Analysis and Discussions

5.1 Detailed Analysis of Computational Efficiency

5.1.1 Pre-Training

The experimental results outlined in Table 4 offer a comprehensive comparison of the SUBLLM and LLaMA models, highlighting the improvements in pre-training speed and reductions in memory usage across various configurations. Specifically, the table illustrates the performance metrics for models with sizes of 0.25B and 1.3B. The results for the 0.25B model were obtained using a single node equipped with 8 A100 GPUs. For 1.3B model, the training speed-up were recorded using four nodes, while the max speed-up results were obtained with a single node.

Speed-Up In the analysis of training speed, the SUBLLM model consistently improves in tokens per GPU per second (TGS) and speed-up ratio as the context window size increases. This shows that SUBLLM is more efficient in larger contexts. Meanwhile, the 1.3B model also shows increased speed-up ratios. Although the increment is slightly less than the 0.25B model, it is likely due to higher communication overhead in multi-node setups. During the training process, the calculation of forward pass and backward propagation is related to the context window. Other calculations including gradient calculation and parameter update have nothing to do with the sequence length. Therefore, as the window length increases, the acceleration effect achieved by subsampling is better, and the acceleration ratio increases.

Max Speed-Up The right section of Table 4 shows the maximum achievable speed-up, where the reduced memory is allocated for a larger batch size to further accelerate the pre-training process. We can see that as the sequence length increases from 2k to 8k, the speed-up effect compared with LLaMA becomes more significant, and more importantly larger than the regular speed-up ratio in the same batch size. Noticed that SUBLLM also gains a higher max speed-up ratio as the language model scales up to 1.3B, where the speed-up ratio gap between SUBLLM and LLaMA becomes larger.

Memory Concerning GPU memory usage, SUBLLM markedly improves upon LLaMA, with the memory savings for the 0.25B model increasing from 6GB in a 2k window to 8GB in an 8k window. This substantial reduction in memory usage with larger window sizes underlines SUBLLM's enhanced processing efficiency. The 1.3B model mirrors this pattern, confirming the model's improved efficiency in more extensive configurations. Overall, SUBLLM not only boosts training speeds but also significantly reduces

Table 4. Detailed analysis of computational efficiency in the pre-training phase, where the max speed-up reallocates the saved memory in pre-training for a larger batch size to further explore the maximum speed-up boundary.

Model Size	Context Window	Model	Speed-Up				Max Speed-Up	
			TGS↑	Ratio↑	Mem (GB)↓	Δ Mem	TGS↑	Ratio↑
0.25B	2k	LLaMA	85,925	×1.00	60.16	-	85,462	×1.00
		SUBLLM	107,260	×1.25	53.76	-6.41	107,859	×1.26
	4k	LLaMA	77,590	×1.00	77.66	-	77,423	×1.00
		SUBLLM	99,209	×1.28	69.03	-8.63	100,425	×1.30
	8k	LLaMA	64,959	×1.00	74.15	-	64,741	×1.00
		SUBLLM	86,227	×1.33	66.03	-8.11	87,261	×1.35
1.3B	2k	LLaMA	18,405	×1.00	72.99	-	20,284	×1.00
		SUBLLM	22,831	×1.24	61.80	-11.19	26,219	×1.29
	4k	LLaMA	16,976	×1.00	65.99	-	18,856	×1.00
		SUBLLM	21,341	×1.26	55.81	-10.18	24,773	×1.31
	8k	LLaMA	15,080	×1.00	65.89	-	16,587	×1.00
		SUBLLM	19,390	×1.29	56.19	-9.70	22,264	×1.34

Table 5. Detailed analysis of computational efficiency in the inference phase. Actual retention means the lowest retention rate of the sequence tokens through the depth of the model in the inference phase.

Context Window	Model	Actual Retention	First Token		Non-First Tokens		Memory	
			Latency (ms)↓	Ratio↑	Speed ↑	Ratio↑	Mem (GB)↓	Δ Mem
2k	LLaMA	-	695.16	×1.00	20.82	×1.00	6.98	-
	SUBLLM	43%	496.66	×1.40	26.71	×1.28	5.63	-1.35
4k	LLaMA	-	2,051.59	×1.00	17.83	×1.00	18.49	-
	SUBLLM	44%	1,410.94	×1.45	24.43	×1.37	17.29	-1.20
8k	LLaMA	-	16,249.11	×1.00	12.38	×1.00	61.05	-
	SUBLLM	44%	9,758.40	×1.67	18.80	×1.52	58.61	-2.44

Table 6. The impact of Adam and ScaledAdam optimizers on the model performance and speed-up in pre-training.

	Adam		ScaledAdam	
	Valid Loss↓	Ratio	Valid Loss↓	Ratio
LLaMA	3.725	-	3.693	-
SUBLLM	3.743	×1.33	3.687	×1.32

the memory footprint, making it especially advantageous in larger configurations. This scalability and efficiency position SUBLLM as an attractive option for environments where optimal performance and effective computational resource management are paramount.

5.1.2 Inference

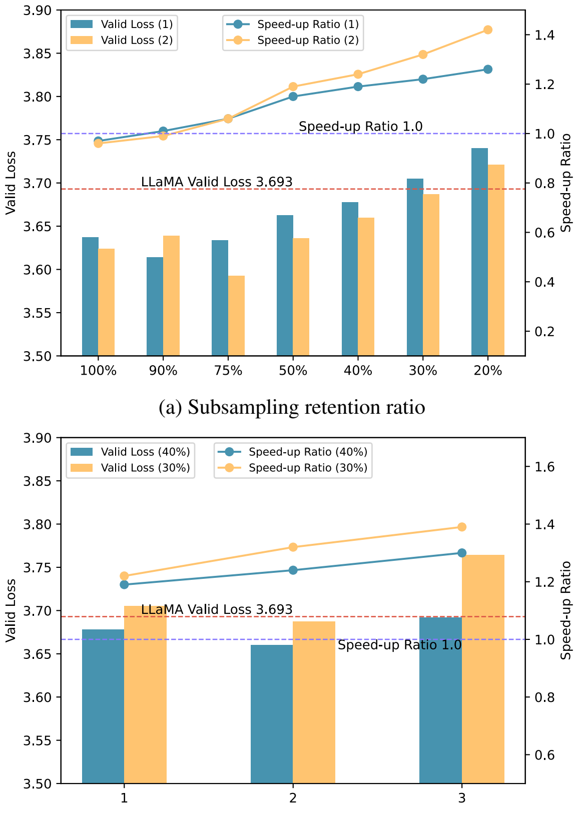
Table 5 provides a detailed analysis of the inference acceleration on one A100 GPU and GPU memory savings for the 1.3B SUBLLM model that subsamples sequences twice to retain 40% of the sequence. The tests covered 1.3B models with 2k, 4k, and 8k window sizes, assessing various performance metrics across corresponding input lengths of 2k, 4k, and 8k. The test samples were taken from the Slimpajama test set, and the inference batch size was set to 8. The metrics evaluated included initial token latency and the acceleration ratio of SUBLLM over LLaMA, non-initial token latency and its acceleration ratio, GPU memory usage during inference, the memory savings achieved by SUBLLM, and the actual token retention ratio during inference.

The results show that the actual token retention ratio during inference slightly exceeds the 40% set during training, which is expected due to the training balancer allowing for a fluctuation range of ±5%. As the input sequence length increases, both the initial and subsequent token decoding speeds show greater acceleration ratios. This improvement is related to model inference with Pytorch. SUBLLM reduces the computational load and significantly decreases the time spent on computations within the CUDA kernels. However the time required to launch CUDA kernels remains constant across different sequence lengths. With longer sequences, the proportion of time spent within CUDA kernels becomes more significant. Thus, these factors leads to higher speed-up ratios for longer sequences.

Additionally, the observed increase in memory savings correlates with the reduction in the size of key-value cache, which refers to the length of the key-value pairs stored in the cache for the attention mechanism during inference. These findings demonstrate that the SUBLLM model structure yields greater benefits when processing longer texts during inference, highlighting its efficiency and effectiveness in large-scale text handling.

5.2 Analysis on Subsampling

We explore the impact of different subsampling setups on the model performance (i.e., valid loss) and training speed-up ratio, including the number of continuous subsampling modules and retention ratio. This retention ratio refers to the lowest retention rate of the original sequence length through the depth of the model. Aiming to search for an optimal configuration with an appropriate speed-up ratio and better performance that can be applied universally, we conduct experiments on the proposed SUBLLM with 0.25B parameters for training efficiency and parameter selection. Note that if the two configurations have close speed-up ratios, we choose the one with better performance as the optimal configuration.



(a) Subsampling retention ratio

(b) The number of subsampling blocks

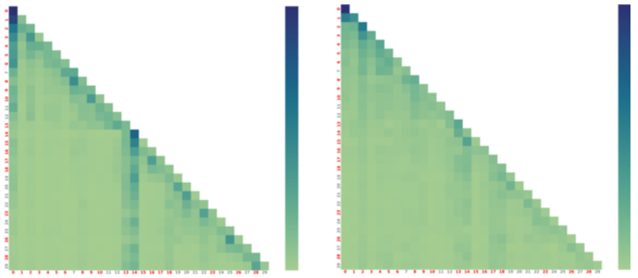
Figure 2. The impact of various subsampling setups on model performance and speed-up in pre-training. Figure 2a illustrates the model with one and two subsampling modules, denoted by (1) and (2), respectively.

Retention Ratio From Figure 2a we can see that SUBLLM achieves the lowest valid loss by retaining 90% of tokens with subsampling once and retaining 75% of tokens with subsampling twice, yet the speed-up ratio is relatively low around 1.0. When the retention ratio is 40% and 30%, the valid loss of SUBLLM is lower than LLaMA and the training speed-up is significant, especially with subsampling twice continuously. In addition, for 100% retention rate without discarding tokens in pre-training, the valid loss of SUBLLM is still lower than LLaMA, demonstrating the effectiveness of the bypass module of SUBLLM for convergence acceleration and loss reduction.

Subsampling Times We further conduct experiments on the variants of subsampling times under the retention ratios of 30% and 40%. As shown in Figure 2b, the valid loss of subsampling twice is lower than that of subsampling once. It can also be observed from the figure that subsampling three times leads to the valid loss increase, especially for a 30% retention ratio. This probably results from that the Transformer blocks are relatively few between paired subsampling modules, which is not sufficient for extracting high-level semantic information for each processed sequence and leads to suboptimal performance. Given the priority of performance optimization, we consider 2 successive subsampling with retaining 40% tokens as the optimal configuration with a prominent pre-training efficiency.

5.3 Analysis of Optimizer

The experimental results presented in Table 6 analyze the impact of different optimizers on the performance of 0.25B models, specifically focusing on Adam and ScaledAdam. Both LLaMA and SUBLLM are evaluated with valid loss and speed-up ratios during pre-training, where the batch sizes are the same between these two models. Employing ScaledAdam for optimization leads to lower valid losses for both models (especially for SUBLLM), suggesting ScaledAdam could facilitate convergence and improve model performance. SUBLLM achieves a consistent speed-up ratio of 1.33 with Adam and 1.32, indicating that ScaledAdam improves model performance while not hurting computational efficiency. This analysis sets a benchmark for future optimizations and model enhancements.



(a) SUBLLM

(b) LLaMA

Figure 3. Attention distribution of the 5th block for SUBLLM and the 6th block for LLaMA, where kept indexes in subsampling are highlighted in red.

5.4 Validity of Subsampling

To analyze the distribution of indexes after subsampling, we examine the attention distribution of the 1.3B SUBLLM model retaining 40% of tokens with two subsampling modules. First, we compare the index distribution after the first subsampling in SUBLLM model with the attention distribution within the pre-subsampling block (the fifth block). As illustrated in Figure 3a, it is evident that most tokens which receive significant attention, visible as distinct vertical stripes in the pre-subsampling attention distribution, are preserved by the subsampling module. Additionally, we analyze attention distribution of the sixth block in the 1.3B LLaMA model, where SUBLLM begins to compute on its shorter sequences after first subsampling. We compare it with the same index retention distribution following the first subsampling module in SUBLLM. In this study, we hypothesize that for language models, the semantics at equivalent depths should be similar. As shown in Figure 3b, it can be observed that the tokens identified as crucial by LLaMA align closely with the subsampled regions where attention is computed in SUBLLM at the same depth, which further indicates effective preservation of important semantic information through the subsampling process.

6 Conclusion

In this study, we propose SUBLLM, a novel network architecture that utilizes text sequence redundancy and token significance to enhance training and decoding speeds while preserving few-shot learning capabilities. SUBLLM features an innovative subsampling mechanism allowing for customizable token retention ratios and includes a bypass module that significantly speeds up model convergence. Our findings indicate that the ScaledAdam optimizer supports this architecture by enhancing its convergence performance. This architecture is compatible with existing optimization methods within the LLaMA model family, ensuring wide applicability. Future research will investigate the impact of sequence compression ratios on SUBLLM to further understand token sequence subsampling, as well as further validate the model's scalability.

References

[1] J. Ainslie, T. Lei, M. de Jong, S. Ontañón, S. Brahma, Y. Zemlyanskiy, et al. Colt5: Faster long-range transformers with conditional computation. In *Proceedings of EMNLP*, pages 5085–5100, 2023.

[2] A. Bapna, N. Arivazhagan, and O. Firat. Controlling computation versus quality for neural sequence models. *arXiv preprint arXiv:2002.07106*, 2020.

[3] F. Barbieri, J. Camacho-Collados, L. E. Anke, and L. Neves. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of EMNLP 2020*, pages 1644–1650, 2020.

[4] A. Botev, S. De, S. L. Smith, A. Fernando, G.-C. Muraru, et al. Recurrentgemma: Moving past transformers for efficient open language models. *arXiv preprint arXiv:2404.07839*, 2024.

[5] T. B. Brown, B. Mann, M. Subbiah, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901, 2020.

[6] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

[7] Z. Dai, G. Lai, Y. Yang, and Q. Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *Advances in Neural Information Processing Systems*, 33:4271–4282, 2020.

[8] T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[9] E. Frantar and D. Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[10] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[11] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[12] Z. He, M. Yang, M. Feng, J. Yin, X. Wang, J. Leng, and Z. Lin. Fourier transformer: Fast long range modeling by removing sequence redundancy with fft operator. In *Findings of ACL 2023*, pages 8954–8966, 2023.

[13] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *Proceedings of ICLR*, 2021.

[14] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024.

[16] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[17] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, et al. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] T. Lei, J. Bai, S. Brahma, J. Ainslie, K. Lee, Y. Zhou, N. Du, et al. Conditional adapters: Parameter-efficient transfer learning with fast inference. 36:8152–8172, 2023.

[19] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[20] X. Ma, X. Yang, W. Xiong, B. Chen, L. Yu, H. Zhang, J. May, L. S. Zettlemoyer, O. Levy, and C. Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *arXiv preprint arXiv:2404.08801*, 2024.

[21] L. C. Magister, J. Mallinson, J. Adámek, E. Malmi, and A. Severyn. Teaching small language models to reason. In *Proceedings of ACL*, pages 1773–1781, 2023.

[22] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, R. Y. Y. Wong, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

[23] P. Nawrot, A. Łańcucki, M. Chochowski, D. Tarjan, and E. M. Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

[24] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. Fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[25] B. Peng, E. Alcaide, Q. Anthony, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[26] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[27] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[28] D. Raposo, S. Ritter, B. A. Richards, T. P. Lillicrap, P. C. Humphreys, and A. Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[29] B. Shen, Z. Lin, D. Zha, W. Liu, J. Luan, B. Wang, and W. Wang. Pruning large language models to intra-module low-rank architecture with transitional activations. In *Findings of ACL 2024*, pages 9781–9793, 2024.

[30] D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, and N. Dey. Slimpajama: A 627b token cleaned and deduplicated version of redpajama. *https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama*, June 2023.

[31] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pages 1631–1642, 2013.

[32] B. Spector and C. Ré. Accelerating llm inference with staged speculative decoding. *arXiv preprint arXiv:2308.04623*, 2023.

[33] J. Su, M. H. M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[34] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024.

[35] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

[36] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

[37] H. Touvron, T. Lavril, G. Izacard, X. Martinet, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[38] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[39] Q. Wang, Y. Yuan, X. Yang, R. Zhang, K. Zhao, W. Liu, J. Luan, D. Povey, and B. Wang. Subllm: A novel efficient architecture with token sequence subsampling for llm. *arXiv preprint arXiv:2406.06571*, 2024. Full version of this paper.

[40] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[41] M. Xia, T. Gao, Z. Zeng, and D. Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2024.

[42] G. Xiao, J. Lin, M. Seznec, et al. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[43] N. Yang, T. Ge, L. Wang, B. Jiao, D. Jiang, L. Yang, R. Majumder, and F. Wei. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*, 2023.

[44] Z. Yao, L. Guo, X. Yang, W. Kang, F. Kuang, Y. Yang, Z. Jin, L. Lin, and D. Povey. Zipformer: A faster and better encoder for automatic speech recognition. *arXiv preprint arXiv:2310.11230*, 2023.

[45] S. Zhai, W. Talbott, N. Srivastava, C. Huang, H. Goh, R. Zhang, and J. M. Susskind. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

[46] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, 28, 2015.

[47] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Re, C. Barrett, Z. Wang, and B. Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2023.

[48] W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*, 2023.

[49] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. A survey on model compression for large language models. *arXiv preprint arXiv:2308.07633*, 2023.