MoDification: Mixture of Depths Made Easy

Chen Zhang♣, Meizhi Zhong♦, Qimeng Wang♥, Xuantao Lu♥, Zheyu Ye♥, Chengqiang Lu♥, Yan Gao♥, Yao Hu♥, Kehai Chen♦, Min Zhang♦, Dawei Song♣[*]
♣Beijing Institute of Technology
♦Harbin Institute of Technology, Shenzhen
♥Xiaohongshu
chenzhang9702@outlook.com

Abstract

Long-context efficiency has recently become a trending topic in serving large language models (LLMs). And mixture of depths (MoD) is proposed as a perfect fit to bring down both latency and memory. In this paper, however, we discover that MoD can barely transform existing LLMs without costly training over an extensive number of tokens. To enable the transformations from any LLMs to MoD ones, we showcase top-k operator in MoD should be promoted to threshold-p operator, and refinement to architecture and data should also be crafted along. All these designs form our method termed MoDification. Through a comprehensive set of experiments covering model scales from 3B to 70B, we exhibit MoDification strikes an excellent balance between efficiency and effectiveness. MoDification can achieve up to ∼1.2× speedup in latency and ∼1.8× reduction in memory compared to original LLMs especially in long-context applications.

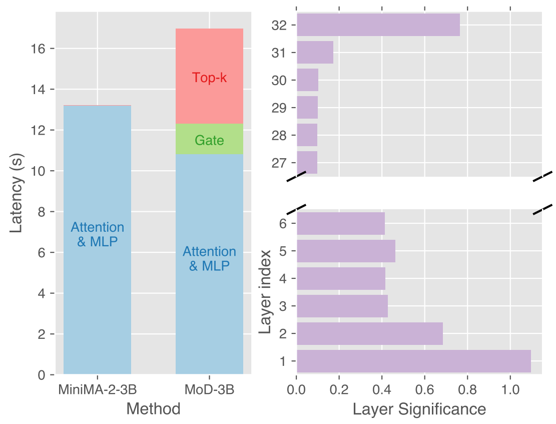

Figure 1: The sub-optimality of MoD due to the use of top-k operator. For efficiency, the efficiency improvement is restricted because top-k operator is not cheap and unimportant computations are preserved by retaining a constant number of tokens at every layer. For effectiveness, the effectiveness is undesired because dominated computations are saved by skipping a constant number of tokens at every layer.

1 Introduction

Long-context efficiency is turning to be one of the core concerns in large language model (LLM) serving (Touvron et al., 2023a,b; Dubey et al., 2024; Bai et al., 2023; Yang et al., 2024). Typically, a long context can incur dramatically huge latency and memory overhead either at prefilling or decoding stage (Wan et al., 2024).

To address this consumption, strategies like speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Liu et al., 2024) and key-value (KV) cache compression (Xiao et al., 2024b; Li et al., 2024; Xiao et al., 2024a) have already been utilized to enhance the latency and memory, respectively. However, dedicating efforts to separate components is not always ideal. As a consequence, mixture of depths (MoD, Raposo et al., 2024) has recently been introduced to simultaneously consider both two facets. Conceptually, the key feature of MoD is enabling LLMs to conditionally eliminate computations of layers over certain tokens. And distinguished from early exiting (EE, Chen et al., 2024), MoD skips over rather than exiting from intermediate layers so that performance is better preserved.

Frustratingly, MoD can hardly guarantee both efficiency and effectiveness without extremely costly training. To be more specific, current advances of MoD are only observed in cases where training is started from scratch, predominately narrowing the impact of MoD.

The sub-optimality is mainly due to the improper use of top-k operator. The top-k operator is basically employed at each layer to determine which tokens should be skipped in the computation of the layer. Unfortunately, for efficiency, the top-k operator is computationally expensive (Wang et al., 2021) and forces a constant number of tokens retained, possibly restricting the efficiency improve-

[*]Dawei Song is the corresponding author.

ment brought by computation saving at one unimportant layer, as shown in Figure 1 (left). For effectiveness, it deviates from the distribution of layer significance (Men et al., 2024), potentially resulting in undesired computation saving at one dominated layer, as shown in Figure 1 (right). These disadvantages largely impede the applications of MoD.

In this paper, in order to broaden the scope of MoD, we target at converting existing LLM checkpoints to MoD ones with our designed MoDification.[1] The designs of MoDification follow two guidelines: 1) alleviating the sub-optimality of MoD, and 2) minimizing the training of MoD. Under the guidelines, we firstly use threshold-p operator as an drop-in replacement of the top-k operator. On one hand, threshold-p operator is not much computationally expensive and allows any number of retained tokens; on the other hand, it can make the layer itself determine the priority of tokens through adaptive gates across layers. We secondly find around 10B tokens with decent diversity are fairly enough in the conversion process.

We conduct experiments on model scales ranging from 3B to 70B. The efficiency and effectiveness of MoDification are verified on such a wide array of scales. With acceptable performance decline, MoDification can achieve up to $\sim$1.2$\times$ speedup in latency and $\sim$1.8$\times$ reduction in memory compared to original LLMs especially in long-context applications. On the contrary, MoD can unexpectedly lead to slow-down in latency under the same experimental settings. Further ablations and discussions validate the trustworthiness of the design choices in MoDification.

2 Background

Long-context Efficiency Over the years, the super expressiveness of large language models (LLMs) has been widely witnessed thanks to the stable transformer architecture (Vaswani et al., 2017) and the massive pretraining (Brown et al., 2020). Concretely, two modules are involved in a transformer layer, namely, self-attention (Attention) module and multi-layer perceptron (MLP) module. Essentially, layer norm (Norm) and residual connection are also armed around. Given the hidden state of the i-th token $\mathbf{x}_i$, the forward process in one transformer layer can usually be depicted as below:

$$\begin{aligned}\mathbf{y}_i &= \text{Attention}(\text{Norm}(\mathbf{x}_i)) + \mathbf{x}_i,\\ \mathbf{z}_i &= \text{MLP}(\text{Norm}(\mathbf{y}_i)) + \mathbf{y}_i,\end{aligned} \tag{1}$$

While the transformer architecture admits the scaling of pretraining, a critical bottleneck during inference is also revealed by the transformer. In other words, the transformer architecture mainly grants parallelism during training, yet only generates tokens one followed by another and requires KV caches adequately stored during inference. This bottleneck may easily slow down the expected speed and shoot over the provided hardware, especially for long-context circumstances (Xiao et al., 2024b).

To this demand, many approaches have been proposed to either counter the latency or the memory increment (Wan et al., 2024). Among these approaches, speculative decoding (Chen et al., 2023) is a representative cluster in reducing the latency and KV cache compression (Xiao et al., 2024b) is another symbolic group in mitigating the memory. However, for an integrated inference system, it is might be somehow redundant to conquer the latency and the memory separately. Consequently, later solutions to some extent shift the focus to conditional computation (Bengio et al., 2015), in which early exiting (EE, Chen et al., 2024) and mixture of depths (MoD, Raposo et al., 2024) are of central roles.

Mixture of Depths In a nutshell, MoD attempts to eliminate unnecessary layer computations over tokens via a token-level gate (Gate) together with a top-k operator (Top-k). In this way, the forward process within one transformer layer is changed as below:

$$\begin{aligned}g_i &= \text{Gate}(\mathbf{x}_i) = \text{sigmoid}(\mathbf{W}\mathbf{x}_i),\\ f_i &= \text{Top-k}(g_i, k),\\ \mathbf{y}_i &= \begin{cases}\text{Attention}(\text{Norm}(\mathbf{x}_i)) + \mathbf{x}_i, & f_i = 1,\\ \mathbf{x}_i, & f_i = 0,\end{cases}\\ \mathbf{z}_i &= \begin{cases}g_i \cdot \text{MLP}(\text{Norm}(\mathbf{y}_i)) + \mathbf{y}_i, & f_i = 1,\\ \mathbf{y}_i, & f_i = 0,\end{cases}\end{aligned} \tag{2}$$

where $g_i \in [0, 1]$ is a sigmoid-normalized score after a linear weight $\mathbf{W}$, and f_i is an indicator of whether the i-th token is exactly ranked among the Top-k tokens regarding its score yielded by the Gate. In addition, MoD is usually applied in an

[1]MoDification is a re-invented compound representing the process of adapting a LLM to a MoD one, following the naming convention of MoEfication (Zhang et al., 2022).

interleaved manner, that is, only one out of two adjacent layers is equipped with MoD.

It is acknowledged that EE is a fine alternative of MoD. Instead of skipping over intermediate layers, EE aggressively exits from intermediate layers and directly ignores later layers, as in Figure 2. Because of this, it is claimed that MoD is more flexible than EE does, and can offer better performance as well.

Despite the superiority of MoD, it does not come without pain. Although up-to-date progress (Raposo et al., 2024) has indeed supported that MoD delivers great success, the training is always launched from scratch with an enormous amount of data. We argue training MoD with an appropriate quantity of data and from publicly released checkpoints should likewise be a pivotal track that is worthy of exploration. And through easing the training, MoD might fully shed its light on related areas.

In our pilot study, we have uncovered sub-optimality pertained to both efficiency and effectiveness of MoD in above lite training setting. Primarily, the sub-optimality is concerned with the improper use of the top-k operator. From the efficiency perspective, the top-k operator is computationally expensive (Wang et al., 2021) and forces a constant number of tokens retained, and can limit computation saving thus efficiency improvement for unimportant layers. From the effectiveness perspective, the top-k operator suffers a divergence from the true distribution of layer significance (Men et al., 2024), and can undesirably result in excessive computation saving for dominated layers. The evidence can be referred to in Figure 1.

3 MoDification

In this work, we put forward such a method, named MoDification, that is aimed at converting existing LLM checkpoints to MoD ones with slight training compute.

To circumvent the sub-optimality of original MoD, two designs are engaged, in which one is associated with architecture and the other is connected with data. The first one is to use threshold-p operator in place of the top-k operator, as shown in Figure 2. threshold-p operator is not much computationally expensive and allows any number of retained tokens. We then instantly discover this threshold-p replacement correlates MoDification to mixture of experts (MoE) (Shazeer et al., 2017; Fedus et al., 2022) and we can empower MoDification with cutting-edge techniques that have been widely used. The second one is to enhance the data diversity and constrain the training to the scale of 10B tokens. We luckily unveil this data scale is fairly enough in practice.

Architecture After applying threshold-p operator to Equation 2, we make a further refinement to the formulation. The formula can now be briefly formed as below:

$$g_i = \text{Gate}(\mathbf{x}_i), \quad f_i = \text{Threshold-p}(g_i, p),$$
$$\mathbf{y}_i = \begin{cases} g_i \cdot \text{Attention}(\text{Norm}(\mathbf{x}_i)) + \mathbf{x}_i, & f_i = 1, \\ \mathbf{x}_i, & f_i = 0, \end{cases}$$
$$\mathbf{z}_i = \begin{cases} g_i \cdot \text{MLP}(\text{Norm}(\mathbf{y}_i)) + \mathbf{y}_i, & f_i = 1, \\ \mathbf{y}_i, & f_i = 0, \end{cases} \tag{3}$$

where the refinement is imposed as also multiplying the gate value g_i to the Attention module. Plus to this, MoDification also follows the interleaved fashion as MoD does.

In fact, the conditional provided by f_i can be interpreted from an MoE view as below:

$$\mathbf{z}_i = \begin{cases} g_i \cdot \text{MLP}(\text{Norm}(\mathbf{y}_i)) + \mathbf{y}_i, & f_i = 1, \\ (1 - g_i) \cdot \text{NoOp}(\mathbf{y}_i) + \mathbf{y}_i, & f_i = 0, \end{cases} \tag{4}$$

where NoOp generally means no operation is carried out and always leads to zero output.

Thereby, MoDification is fundamentally a two-expert MoE (Fedus et al., 2022) where one of the two experts is NoOp. In this sense, the p value of the threshold-p operator should theoretically be 0.5, which is henceforth leveraged unless otherwise specified. Moreover, to promote the sparsity of MoDification, we are inspired by the expert load balancing technique (Fedus et al., 2022) in MoE and suggest a layer load reducing objective as below:

$$F_j = \mathbb{E}_{\mathbf{x}_i}[f_i], \quad G_j = \mathbb{E}_{\mathbf{x}_i}[g_i],$$
$$\mathcal{R} = \alpha \cdot \sum_{j=1}^{L} F_j \cdot G_j, \tag{5}$$

where α is a coefficient that should be manually tuned, and subscript j denotes the j-th layer among overall L layers. While F_j depicts the fraction of tokens dispatched to the layer, G_j describes the fraction of the routing probability to the layer. And reducing R principally triggers more tokens be assigned to the NoOp and thus skipped by the layer. This reducing objective, together with the language

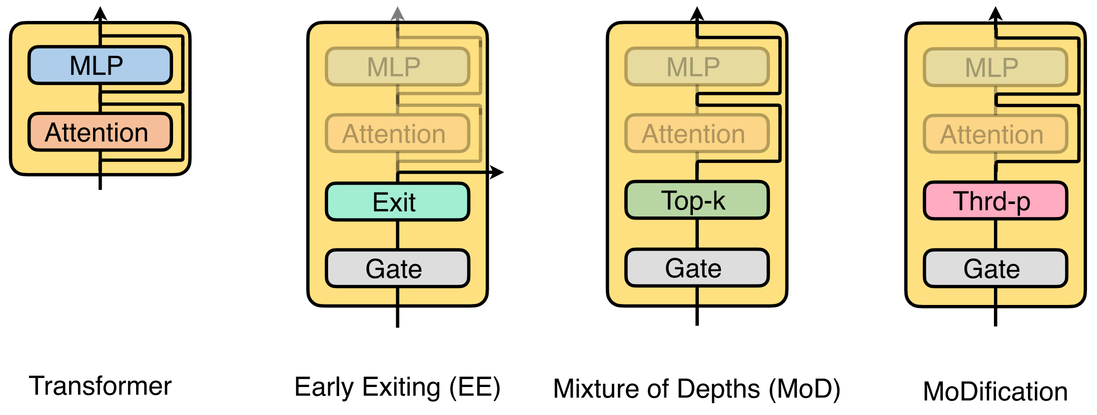

Figure 2: The comparison among transformer, early exiting (EE), mixture of depths (MoD), and our MoDification.

modeling objective $\mathcal{L}$ that drives necessarily layer load preserving, makes MoDification a reasonable game where LLMs learn to strike a good balance between efficiency and effectiveness.

Table 1: The data mixture used to conduct MoDification.

Data	Tokens	Proportion
CCI (2023)	1.5B	14.3%
Wikipedia (2019)	0.2B	1.9%
BookCorpus (2015)	0.1B	1.0%
Cosmopedia (2024)	1.0B	9.5%
FineWeb-Edu (2024)	4.5B	42.9%
RedPajama-Stack (2023)	0.5B	4.8%
RedPajama-GitHub (2023)	1.3B	12.3%
Proof-Pile (2023)	1.3B	12.3%
FLAN (2024)	0.1B	1.0%
Total	10.5B	100.0%

Data With the aim of reducing training time, we try to incorporate diverse domains of data to compensate limited numbers of data points. The data mixture can be found in Table 1.

The proposed data mixture generally covers domains like webs, wikis, books, codes, maths, etc. In doing so, we expect MoDification could be trained to approximate original mixtures of any existing LLM checkpoints.

4 Experiments

Setup We benchmark MoDification mainly against MoD, and also compare these two to original LLM. We conduct evaluation upon MiniMA-2-3B (Zhang et al., 2023a) and LLaMA-2-{7,13,70}B (Touvron et al., 2023b) to examine the scalability of MoDification. For efficiency measures, we consider average latency in second and memory in gigabyte across a wide array of prefilling and decoding lengths ranging from 64 to 2,048 as key metrics. For effectiveness measures, we test above methods on commonly cared datasets including MMLU (Hendrycks et al., 2021), CE-val (Huang et al., 2023), DROP (Dua et al., 2019), BBH (Suzgun et al., 2023), HumanEval (Chen et al., 2021), and GSM8k (Cobbe et al., 2021). The core metrics reported on these datasets are either accuracy, exact matching score, or pass@1 according to the designs.

Implementation We train MoDification and MoD with an overall batch size of 1,024 and a sequence length of 4,096. This leads to 4M tokens per optimization step. The learning rate is 3e-5, and weight decay is 1e-1. The learning rate is scheduled with a linear warm-up and a cosine decay, where the warm-up stage takes 1% optimization steps and the decay stage takes all the left optimization steps. The gradients will be properly clipped to keep the norm under 5e-1. The training precision is bfloat16. DeepSpeed (Rasley et al., 2020), FlashAttention (Dao et al., 2022), and gradient checkpointing (Chen et al., 2024) are necessarily enabled to make the training possible with 16 Nvidia A100 GPUs.

The k for the top-k operator is 512 by default, and the p for the threshold-p operator is 0.5 for MiniMA-2-3B as mentioned earlier but 0.55 for LLaMA-2-{7,13,70}B since slightly larger threshold is demanded for larger LLMs in our trials. The best coefficient α for layer load reducing objective is 0.01 in our developments.

Results From the main comparison results between MoD and MoDification in Table 2, we unearth that MoDification is a way better choice than MoD does. For efficiency, MoDification can reduce both the latency and memory consumption while

Table 2: The main comparison results concerning both efficiency and effectiveness. Better results among the ones reported from MoD and MoDification are **boldfaced** at varying blocks. Results enabled with INT8 quantization (Dettmers et al., 2022) are marked with †.

Method	**Latency** Second	**Memory** GiB	**MMLU** 5-shot Acc	**CEval** 5-shot Acc	**DROP** 3-shot EM	**BBH** 3-shot EM	**HumanEval** 0-shot Pass@1	**GSM8k** 8-shot CoT EM
MiniMA-2-3B	11.0	7.4	40.1	44.7	23.1	31.4	8.9	14.6
- MoD	16.2	**6.7**	25.8	25.7	14.0	27.7	0.0	2.2
- MoDification	**10.9**	**6.7**	**30.0**	**32.2**	**20.5**	**30.8**	**9.8**	**5.7**
LLaMA-2-7B	14.5	16.2	45.0	33.1	33.1	32.1	13.4	12.1
- MoD	21.6	14.9	25.5	26.0	1.8	7.0	0.0	0.0
- MoDification	**13.5**	**14.7**	**36.4**	**28.6**	**25.8**	**30.0**	**8.9**	**10.3**
LLaMA-2-13B	19.4	30.3	54.4	41.2	43.4	38.1	14.0	24.1
- MoD	30.9	28.2	25.7	24.6	1.5	7.8	0.0	0.0
- MoDification	**18.1**	**27.9**	**44.4**	**37.3**	**34.2**	**34.1**	**11.0**	**21.2**
LLaMA-2-70B	201†	72†	68.6	53.9	63.3	51.6	28.7	53.4
- MoD	276†	71†	41.2	30.1	2.5	12.2	0.0	0.0
- MoDification	178†	71†	**54.6**	**46.9**	**43.6**	**38.3**	**15.2**	**38.0**

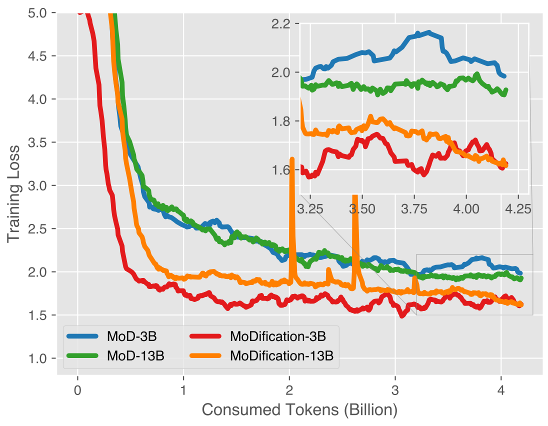

Figure 3: The training losses. The steady loss curves indicate that both MoD and MoDification are rigorously optimized.

Table 3: The number of activated tokens. For example, the number of activated tokens in LLaMA-2-7B is calculated as 32 (layers) * 4,096 (context length) = 131,072. The smaller amounts of activated compute consumed by MoD than that by MoDification shows that MoD is hard to optimize.

Method	***k* / *p***	**Activated Tokens**	**MMLU** 5-shot Acc
LLaMA-2-7B	-	131,072	45.0
- MoD	512	73,728	25.5
- MoD	1,024	81,920	26.2
- MoD	2,048	98,304	25.7
- MoDification	0.55	88,473	36.4

MoD can unexpectedly increase the latency. For effectiveness, MoDification also enjoys better results and and preserves most performance from original considered LLMs. The comparison showcases the superiority of MoDification over MoD. And the realized efficiency merits seem to be positively correlated to the model scales in practice.

One may think that MoD is not properly configured in training and thus shows very unstable and almost meaningless results. To demonstrate that the training of MoD does not suffer from fluctuations such like loss spikes, we hereby display the training losses as in Figure 3. From the loss curves, we conclude that both MoD and MoDification are rigorously optimized. Besides, one may argue that the sparsity of MoD influenced by k does not align with the sparsity of MoDification influenced by p, thus exhibiting unreasonable results. To show that the sparsity of MoD is not the key factor, we confirm in Table 3 that MoD is still hard to optimize even when the number of activated tokens in MoD is smaller than that in MoDification.

To further enhance the plausibility of the comparison, we also compare MoDification to an alternative efficient method named ShortGPT (Men et al., 2024) that is designed to directly drop unimportant layers, therefore decreasing latency and memory, as in Figure 4. The considered ShortGPT baseline is configured as preserving 22 out of 32 layers (i.e., number of activated tokens is 90,112). The major conceptual difference between MoDification and ShortGPT lies in that the layer skipping of ShortGPT happens at task level. We observe that ShortGPT may lead to more apparent efficiency gains, nevertheless, it results in unacceptable performance degradation. This phenomenon implies that MoDification has considerable advantages over other efficient designs.

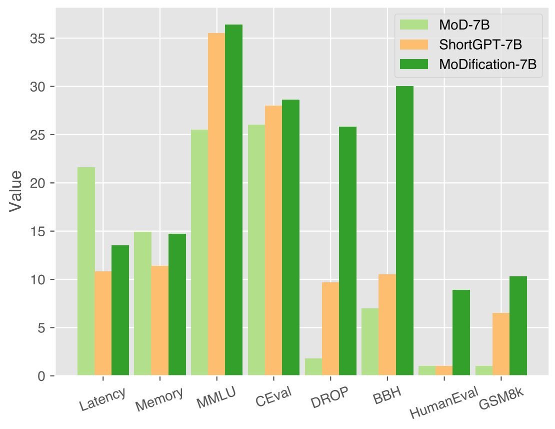

Figure 4: The comparison results concerning other efficient designs. ShortGPT is more efficient yet less effective.

Ablation Studies To get an in-depth understanding of the inner working mechanisms of MoDification, we perform a series of ablation studies on the threshold value p, layer loading reducing coefficient α, etc.

THRESHOLD VALUE p. We show the impact of the threshold value p in Figure 5, from which we inspect that while the memory consumption is always reduced across all threshold values, the latency and the performance can vary a lot from one threshold to another. Disappointingly, though smaller threshold values can cut down memory consumption, they can cause increased latency as a result of limited computation saving. For MiniMA-2-3B in the plot, the most suitable p value is 0.5 regarding the trade-off between efficiency and effectiveness. However, as aforementioned, larger models such as LLaMA-2-70B urge larger p values like 0.55 due to potentially larger structural sparsity.

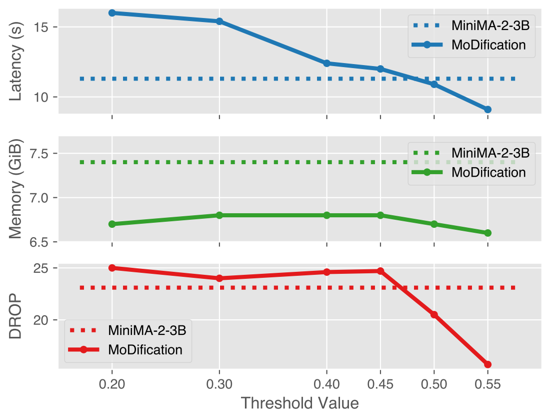

Figure 5: The impact of threshold value p.

LAYER LOAD REDUCING COEFFICIENT α. In Table 4, we know that the layer load reducing coefficient α impacts both efficiency and effectiveness. Particularly, the coefficient trades effectiveness for efficiency. For instance, as α increases itself by an order of magnitude, the latency diminishes, so is the performance. In contrast, without the layer load reducing objective (i.e., when the coefficient is 0), MoDification tends to save very few computations. In this case, the latency is even enlarged. In summary, we assert that α would be better set to 0.01 to reach equilibrium.

Table 4: The impact of layer load reducing coefficient α.

Method	Latency	Memory	GSM8k
	Second	GiB	8-shot CoT EM
MiniMA-2-3B	11.0	7.4	14.6
- $\alpha = 0$	15.7	6.7	8.3
- $\alpha = 0.001$	14.4	6.6	7.0
- $\alpha = 0.01$	10.9	6.7	5.7
- $\alpha = 0.1$	8.2	6.4	1.7

INTERLEAVED V.S. FULL STRATEGY In our design, we follow the de facto strategy of original MoD and use MoDification in an interleaved style where MoDification is applied at every other layer. To confirm the interleaved strategy is the best choice, we compare the interleave startegy to a tactic that applies MoDification to top half layers and another that applies MoDification to full layers, as in Table 5. We note that the interleaved strategy is better than the half strategy in efficiency and is better than the full strategy in effectiveness. The reasons sit behind may be that the half strategy is inclined to conservatively skip too few computations and the full strategy is likely to skip overly many computations and affect the performance. To sum up, the interleaved strategy is currently the recommended option.

Table 5: The impact of interleaved strategy.

Method	Latency	Memory	MMLU
	Second	GiB	5-shot Acc
Interleave	10.9	6.7	30.0
Half	12.8	6.7	37.4
Full	10.5	6.5	27.4

SHARED V.S. SEPARATE GATE. We also include a refinement in MoDification where a shared gate value is multiplied to both the Attention and MLP modules if they are not skipped. Hereby, we would like to test whether two separate gate values respectively for the Attention and MLP modules are better or not. We detect from Table 6 that the performance would not be boosted by the separate gates but the latency would be elevated by the separate gates since additional gating is introduced. So we believe the shared gate is a fair pick.

Table 6: The impact of shared gate.

Method	**Latency**	**Memory**	**BBH**
	Second	GiB	3-shot Acc
Shared	10.9	6.7	30.8
Separate	12.0	6.7	30.6

FULL V.S. GATE TUNING. In our design, we tune all parameters including the original parameters from the LLM and the additional parameters from the Gate. However, we can also alternatively tune the parameters from the Gate only. We have observed significant performance detriment by gate tuning, against full tuning. For instance, on GSM8k, the score drops from 10.3 to 6.5 for MoDification-7B.

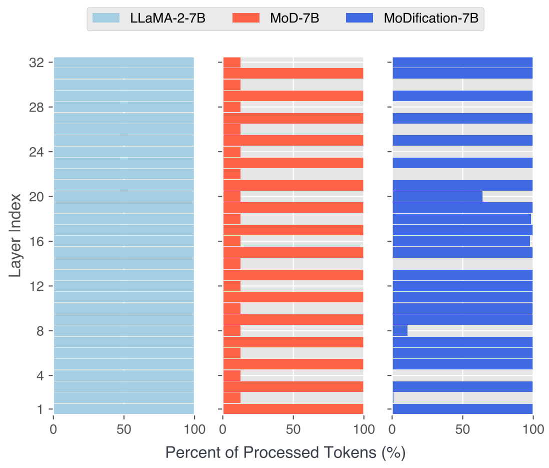

Figure 6: The visualization of executed computations of layers over tokens. MoD can save computations to a certain degree. MoDification can save computations in a more flexible way.

Execution Visualization For a more intuitive comparison between MoD and MoDification, we attempt to clearly illustrate why MoDification is more promising than MoD via visualizations of executed layers over input tokens correspondingly for MoD and MoDification.

In Figure 6, we unleash that MoD saves computations with the sub-optimality due to the use of top-k operator. The forced constraint that a constant number of tokens should be retained may: 1) limit the computation saving (efficiency) when computations of more tokens should be skipped, 2) affect the computation preserving (effectiveness) when computations of fewer tokens should be skipped. Conversely, MoDification permits more flexible computation savings, leading to a better balance between efficiency and effectiveness.

Latency Profiling We previously only provide latency results in average values, and here we would like to share more latency results across all prefilling lengths and accordingly decoding lengths to offer detailed insights on how MoDification overwhelms MoD.

We should tell from Figure 7 that MoDification surpasses MoD at arbitrarily any prefilling length or decoding length. And the micro gains add up to a macro gain in average. Besides, MoDification regularly overtakes LLaMA-2 in efficiency. We also find the latency is more sensitive to the decoding length increment than the prefilling length increment. Explicitly, the latency approximately doubles when the decoding length doubles but only increases a bit when the prefilling length doubles. We conjecture the distinction is resulted from that prefilling is parallel while decoding is sequential. And this hints that future explorations should be emphasized on decoding-time efficiency. Trickily, we also uncover that the latency delta along the increment of prefilling length gradually becomes larger. This pinpoints that prefilling-time efficiency would possibly be a concern when the prefilling length is immense, say more than 10k. This naturally drives us to explore the potential of MoDification in long-context applications later in Section 5.

Latency Decomposition More precisely, we investigate the latency through a nano view. To put it differently, we decompose the latency to three parts: the one consumed by Attention and MLP, by Gate, and by Top-k or Threshold-p. Furthermore, the Threshold-p part could be broken down to: Branch phase where some tokens are selected and the others are skipped, Merge phase where both selected and skipped tokens are merged after. The resultant chart is positioned in Figure 8.

It is manifested that MoDification not only yields smaller latency from the Thrd-p, but also yields smaller latency from the Attention and MLP. This denotes that MoDification benefits from both the transition from the Top-k to the Thrd-p and the removal of hard constraint that at least k tokens should be retained.

It is questionable why MoDification can have

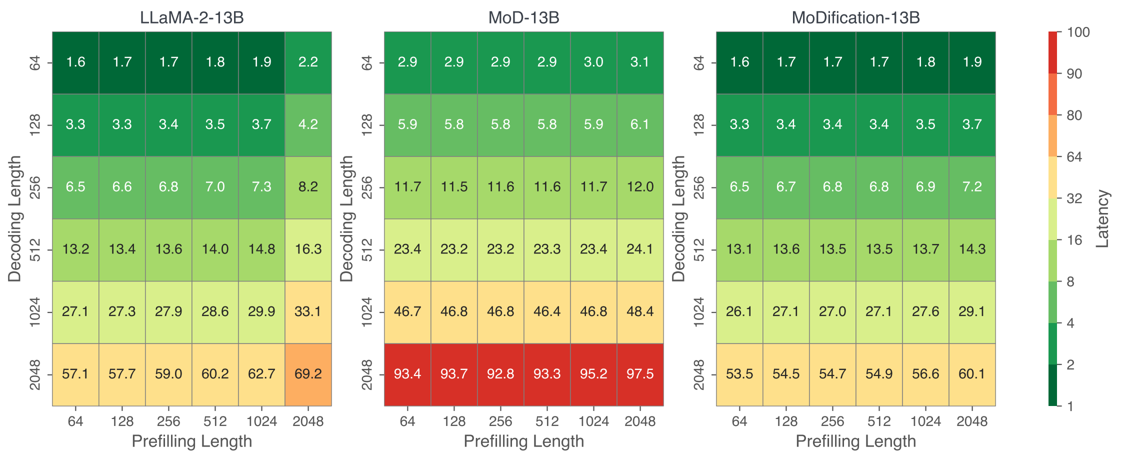

Figure 7: The detailed latency results across different prefilling lengths and decoding lengths. MoDification overwhelms both MoD and LLaMA-2 at any prefilling or decoding length.

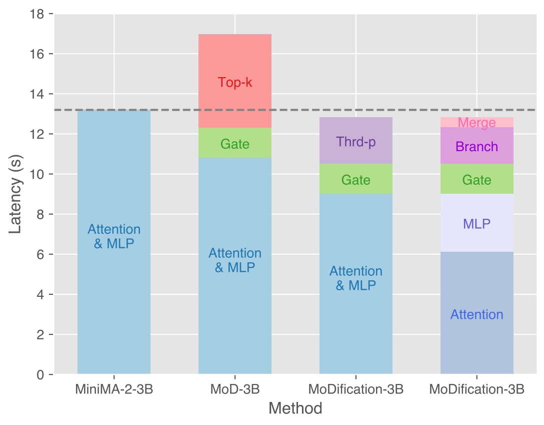

Figure 8: The nano view towards the latency.

Table 7: The exemplary latency results across prefilling and decoding stages. Prefilling latency and per-token decoding latency are reported for prefilling and decoding stages, respectively, in milliseconds and upon MiniMA-2-3B.

Method	Stage	Attention & MLP	Gate	Top-k / Thrd-p
MoD	prefill	29.9	3.1	9.8
MoDification	prefill	24.3	3.1	4.1
MoD	decode	20.9	2.7	4.9
MoDification	decode	18.8	2.7	3.9

smaller latency. Intuitively, the computationally expensive top-k operator only dominates prefilling stage and is computationally comparable to threshold-p operator at decoding stage, while prefilling latency is not the main concern in comparison to cumulatively decoding latency as in Figure 7. However, the above conjecture largely neglects the superiority of the non-uniformity yielded by threshold-p operator as in Figure 6. To counter this, we showcase exemplary latency results across prefilling and decoding stages in Table 7.

For Gate latency, MoDification is nothing different from MoD, and decoding one is smaller than prefilling one as decoding only processes one token at one time. For Top-k / Thrd-p latency, MoDification is significantly faster than MoD at prefilling stage due to the transition from top-k operator to threshold-p operator. Further, MoDification is comparable with MoD at decoding stage.

For Attention and MLP latency, MoDification would skip most tokens at one unimportant layer, significantly reducing latency compared to constantly retain tokens from MoD; MoDification would retain most tokens at one important layer, marginally yielding latency increment compared to MoD. The true reason sits behind may be that decoding is memory-bounded instead of compute-bounded (Yuan et al., 2024), so the latency will increase significantly when the number of retained tokens excels a minimal number, then increase slowly until the number of retained tokens goes beyond an acceptable range regarding memory bandwidth.

5 Long-context Applications

While we have fully justified the usefulness of MoDification in comparably short-context scenarios, we are obliged to substantiate the potential of MoDification in long-context applications. We take two long-context applications as testbeds, i.e., long-text generation that requires long decoding and long-context retrieval that requires long prefilling. As a prerequisite, we extend the context capability of MoDification-7B from 4k to 32k. We adopt NTK-aware interpolation (Rozière et al., 2023) and conduct post training on PG19 dataset (Rae et al., 2019).

In long-text generation scenario where prompts sampled from Proof-Pile (Azerbayev et al., 2023), MoDification-7B can achieve up to $\sim 1.2\times$ speedup in latency and $\sim 1.8\times$ reduction in memory in comparison to LLaMA-2-7B. And through long-context extension, MoDification is compatible with long-context modeling and consistently behaves in low perplexity without compromise on efficiency.

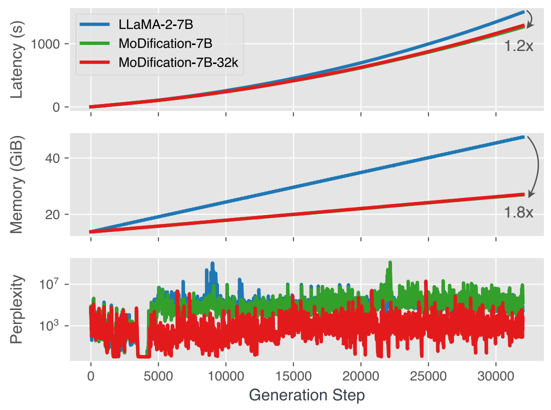

Figure 9: The application to long-text generation. MoDification-7B can achieve up to $\sim 1.2\times$ speedup in latency and $\sim 1.8\times$ reduction in memory in in comparison to LLaMA-2-7B. And MoDification is compatible with long-context modeling.

In long-context retrieval scenario where questions are gathered from Qasper (Dasigi et al., 2021), MoDification-7B-32k yields 10.3 on Qasper, which is competitive with 10.2 from LLaMA-2-7B-32k. In contrast, StreamingLLM-7B (Xiao et al., 2024b) which significantly prunes KV caches yields a dramatically degraded number of 6.9.

6 Related Work

Efficient Language Models Improving efficiency of language models is a long-standing and challenging task (Wan et al., 2024). Besides system-level efficiency optimizations, algorithm-level efficiency optimizations mainly lie in model compression and efficient architecture. Model compression approaches (Bucila et al., 2006) aim to reduce the model size directly. These compression approaches can be grouped into quantization (Dettmers et al., 2022), pruning (Ma et al., 2023), and distillation (Yang et al., 2022; Zhang et al., 2024b,a, 2023b). Efficient architectures (Child et al., 2019; Shazeer, 2019) instead aim to reduce the quadratic time complexity of attention to quasi-linear one. Mixture of depths (Raposo et al., 2024) stands at the intersection of model compression and efficient architecture. On one hand, mixture of depths can skip layers, which can be viewed as a type of pruning. On the other hand, mixture of depths can drop tokens, which can be viewed as a sort of complexity reduction in time.

Conditional Computation A third perspective towards improving efficiency of language models would be conditional computation (Bengio et al., 2015). It aims to relieve unnecessary computations by conditionals. This dynamic property can to the maximum extend preserve effectiveness while improving efficiency. Typical work in this area would be mixture of experts (Shazeer et al., 2017; Fedus et al., 2022), early exiting (Chen et al., 2024), and mixture of depths. And mixture of depths is recently viewed as one of the best choices among available conditional computation approaches.

7 Conclusions

It is no doubt that MoD is an ideal choice to lift the efficiency of LLMs. In this paper, we would like to address the concern that MoD comes with a must that costly training (from scratch) should be executed. Via both architecture and data designs, we make it possible to convert existing LLMs to MoD ones with appropriate data. The designed MoDification successfully outweighs MoD in both efficiency and effectiveness.

Limitations

Our work is limited in the following two aspects: 1) we do not apply MoDification to recently released LLMs such like LLaMA-3 (Dubey et al., 2024) or Qwen-2 (Yang et al., 2024), 2) we do not apply MoDification to extremely lengthy texts, say more than 100k (Zhang et al., 2024c), and 3) we have to admit the uniform architecture of MoD might be more efficient on special hardware platforms.

Acknowledgments

This work is funded in part by the Natural Science Foundation of China (grant no: 62376027).

References

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck.

2023. Llemma: An open language model for mathematics. *Preprint*, arXiv:2310.10631.

BAAI. 2023. Cci, chinese corpora internet.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *CoRR*, abs/2309.16609.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. 2024. Cosmopedia.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 535–541. ACM.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. EE-LLM: large-scale training and inference of early-exit large language models with 3d parallelism. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25:70:1–70:53.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *CoRR*, abs/2208.07339.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2368–2378. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: LLM knows what you are looking for before generation. *CoRR*, abs/2404.14469.

Zhuorui Liu, Chen Zhang, and Dawei Song. 2024. How speculative can speculative decoding be? In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 8265–8275. ELRA and ICCL.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. 2024. Fineweb-edu.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *CoRR*, abs/2403.03853.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

David Raposo, Samuel Ritter, Blake A. Richards, Timothy P. Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *CoRR*, abs/2404.02258.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506. ACM.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open foundation models for code. *Preprint*, arXiv:2308.12950. ArXiv: 2308.12950.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. Efficient large language models: A survey. *Trans. Mach. Learn. Res.*, 2024.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*, pages 97–110. IEEE.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024a. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *Preprint*, arXiv:2410.10819.

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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *CoRR*, abs/2407.10671.

Yi Yang, Chen Zhang, and Dawei Song. 2022. Sparse teachers can be dense with knowledge. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3904–3915. Association for Computational Linguistics.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM inference unveiled: Survey and roofline model insights. *CoRR*, abs/2402.16363.

Chen Zhang, Dawei Song, Zheyu Ye, and Yan Gao. 2023a. Towards the law of capacity gap in distilling language models. *CoRR*, abs/2311.07052.

Chen Zhang, Yang Yang, Qiuchi Li, Jingang Wang, and Dawei Song. 2024a. Task-agnostic distillation of encoder-decoder language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 15629–15639. ELRA and ICCL.

Chen Zhang, Yang Yang, Jiahao Liu, Jingang Wang, Yunsen Xian, Benyou Wang, and Dawei Song. 2023b. Lifting the curse of capacity gap in distilling language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4535–4553. Association for Computational Linguistics.

Chen Zhang, Yang Yang, Qifan Wang, Jiahao Liu, Jingang Wang, Wei Wu, and Dawei Song. 2024b. Minimal distillation schedule for extreme language model compression. In *Findings of the Association for Computational Linguistics: EACL 2024, St. Julian's, Malta, March 17-22, 2024*, pages 1378–1394. Association for Computational Linguistics.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024c. ınftybench: Extending long context evaluation beyond 100k tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15262–15277. Association for Computational Linguistics.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 877–890. Association for Computational Linguistics.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27. IEEE Computer Society.