Leveraging the True Depth of LLMs

Anonymous authors
Paper under double-blind review

Abstract

The remarkable capabilities of Large Language Models (LLMs) are shadowed by their immense computational cost. While recent work has shown that many LLM layers can be reordered or even removed with minimal impact on accuracy, these insights have not been translated into significant inference speedups. To bridge this gap, we introduce a novel method that restructures the computational graph by grouping and evaluating consecutive layer pairs in parallel. This approach, requiring no retraining, boosts inference throughput by 1.05x–1.20x while maintaining 95-99% of the original model's accuracy on standard benchmarks. We demonstrate the practical value of this method for large-scale LLM deployment and show that some of the accuracy trade-off can be recovered with lightweight fine-tuning of the parallelized layers.

1 Introduction

The rapid advancement of LLMs has revolutionized Artificial Intelligence applications across industries. However, the ever-increasing computational demands of these models, with parameters often numbering hundreds of billions, present significant commercial challenges. Efficient inference is crucial for organizations that deploy these models at scale, as it directly impacts operational costs, user experience, and environmental sustainability (Singh et al., 2025; Xu et al., 2024; Wu et al., 2022). Monthly cloud computing expenses for LLM inference can reach millions of dollars for high-traffic applications, making optimization techniques essential. In addition, reducing inference latency is critical for real-time applications and for deploying models on devices with more limited compute.

Thus, the development and implementation of efficient inference methods has become a key differentiator in the competitive AI market, driving both innovation and profitability.

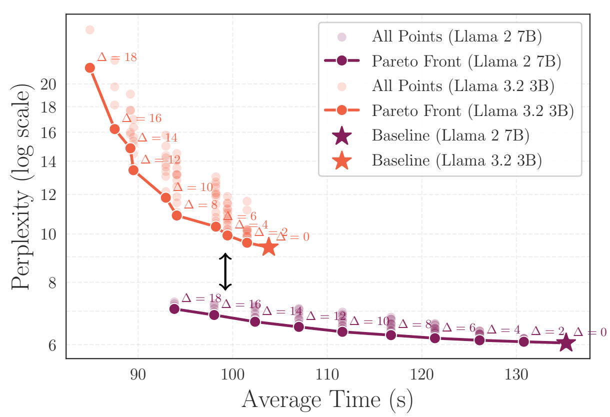


Figure 1: The effect of LP on execution time (4K tokens) and perplexity (measured against RedPajama (Together Computer, 2023)). The bigger Llama model achieves both better perplexity and faster execution speeds than the smaller Llama model.

LLMs have evolved to incorporate architectures with hundreds of layers (OpenAI, 2023; Grattafiori et al., 2024). These models are constructed from stacked transformer blocks, each comprising attention and feed-forward subblocks, with a residual stream traversing the entire architecture to facilitate efficient gradient propagation during training. This architectural choice parallels the design principles of ResNets (He et al., 2015), where research has shown that network depth may be partially redundant, allowing layer reordering without significant performance loss (Veit et al., 2016). Recent investigations have revealed similar flexibility in transformer architectures (Lad et al., 2024), where interventions such as layer removal and swapping are applied without large performance degradations. Although these findings challenge our understanding of LLMs' true effective depth, their potential for optimizing inference efficiency remains unexplored.

Inspired by this observed layer independence, we investigated several interventions to the computational graph of pre-trained LLMs. Our exploration of layer shuffling, pruning, and merging revealed that multiple consecutive block pairs can be processed in parallel while maintaining accuracy across perplexity and In-Context Learning (ICL) benchmarks. This led us to propose Layer Parallelism (LP), a novel approach that enhances inference speed when performing inference in the Tensor Parallel (TP) regime. LP modifies the computational graph of a pre-trained LLM to reduce the inter-device communication by half, with a minimal drop in model performance. Furthermore, we show that this performance degradation can be partially mitigated through targeted fine-tuning procedures.

Contributions. Our contributions can be summarized as follows:

- We explore the space of interventions on a pre-trained LLM layers and find that some transformations, such as contiguous parallelization, preserve model performance
- We find that we can define a parallelization transform on the computational graph of two sequential Transformer layers, and stack this parallelization operation to several sequential pairs of layers without losing significant ICL performance. Our approach, which we call LP, can be applied to existing Transformer models.
- We show that by fine-tuning the LP blocks we can recover some of the lost performance, while retaining the previously obtained speed-up.

2 Related work

The effective depth of Deep Networks. Theoretically, given enough width, any feed-forward network with at least one hidden layer can model any function (Pinkus, 1999). In practice, it is easier to achieve high expressivity by increasing the model's depth. However, naively increasing network depth can complicate optimization, since the gradients now have to flow through many layers. To alleviate this problem, ResNets (He et al., 2015) introduced skip connections at regular intervals to allow an easy flow of the gradient to the first layers. Alternatively, Inception (Szegedy et al., 2014) explored approaches to boost computational power by adding additional processing units along different parallel pathways in the computational network, rather than just along a single sequential path. A unification of both methods can be found in the Highway Networks (Srivastava et al., 2015), where the skip connection of the residual blocks consists of another block of compute. Nowadays, residual connections are ubiquitous in large models.

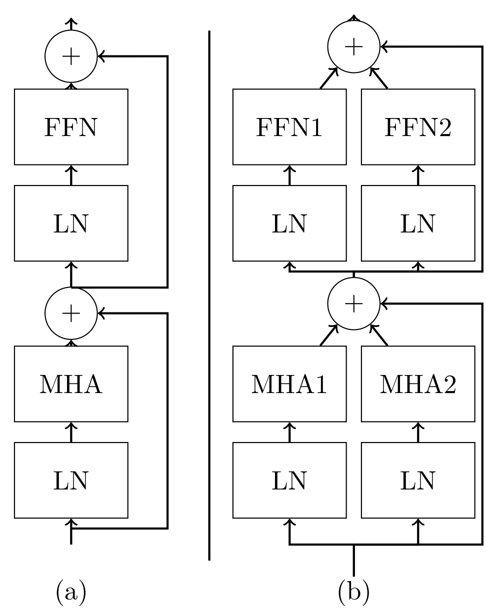


Figure 2: Comparison of a normal transformer block (a) with our layer parallel implementation (b). Divergent paths in (b) are split across the Tensor Parallel axis.

Efficient inference of LLMs. Several complementary approaches exist for enhancing the computational efficiency of large-scale models, primarily through pruning/sparsity, quantization, and parallelism. Pruning (LeCun et al., 1989; Han et al., 2015; 2016; Frantar & Alistarh, 2023) constitutes a dimensional reduction methodology that systematically eliminates redundant parameters while preserving model performance, thereby introducing architectural sparsity. This methodology is founded on empirical evidence demonstrating that neural networks frequently exhibit overparameterization, containing numerous weights with negligible contributions to the output. Through sophisticated pruning strategies, the inherent sparsity support in contemporary accelerators can be leveraged to enhance both memory utilization and computational efficiency (Zhang et al., 2020; Wang et al., 2021). Early-exit (Teerapittayanon et al., 2017; Zhou et al., 2020) can be seen as a way of runtime layer-wise pruning, which halts the LLM forward pass when the next token certainty is high in the intermediate layers. This approach can be also thought as a way of reducing the effective depth of the model at test time. In contrast, quantization encompasses the transformation of floating-point numerical representations (predominantly FP32) into reduced-precision integer formats, such as INT8 or INT4 (Shen et al., 2019; Han et al., 2016; Jacob et al., 2018). When implemented on hardware accelerators, these lower-precision representations allow for higher FLOPs and better use of the memory bandwidth, addressing a primary bottleneck in modern large-scale models (Gholami et al., 2024); moreover, integer-based computations yield enhanced processing speed and substantially improved energy efficiency (Horowitz, 2014). Finally, parallelization techniques during inference, such as tensor and pipeline parallelism, enable the distribution of computational workload across multiple accelerators, thereby reducing latency and increasing throughput, although this often requires careful consideration of communication overhead and load balancing (Li et al., 2024; Narayanan et al., 2021).

Parallel attention-feedforward fusion. GPT-J (Wang & Komatsuzaki, 2021) introduced a parallel formulation of the transformer decoder layer, executing attention and feedforward sub-blocks concurrently:

$$y = x + \mathrm{MHA}(\mathrm{LayerNorm}(x)) + \mathrm{MLP}(\mathrm{LayerNorm}(x))$$

For models trained with tensor parallelism, this modification halves the number of required all-reduce operations. Additionally, it reduces memory bandwidth usage by eliminating one read and write operation of hidden states from HBM. The input projections for the attention and MLP sub-blocks can also be fused into a single kernel, further increasing arithmetic density. Consequently, training time is reduced by approximately 15% without observable performance degradation. PaLM (Chowdhery et al., 2023) also adopted this formulation, noting that negative effects from deviating from the standard transformer self-attention diminish with increasing model size. This parallel formulation continues to be employed in more recent LLMs, including Gemini 1.5 Flash (Georgiev et al., 2024).

In contrast to these methods, which require training a new model from scratch with a modified architecture, our approach is applied post-hoc to already-trained models (Touvron et al., 2023; Grattafiori et al., 2024; Yang et al., 2025). This highlights two other key differences. First, the granularity of parallelism differs: GPT-J-style models parallelize the attention and feed-forward sub-blocks *within* a single layer, whereas our method parallelizes *entire consecutive layers*, directly reducing the model's effective depth. Second, LP accepts a trade-off by approximating the original computation, which results in a slight performance degradation in exchange for inference acceleration. This degradation can be largely recovered with light fine-tuning. The GPT-J architecture, by contrast, is exact by definition, as the model was trained with it from the beginning.

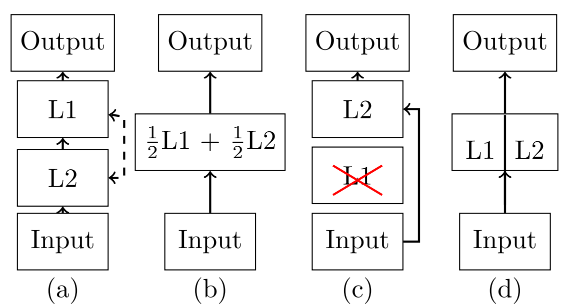
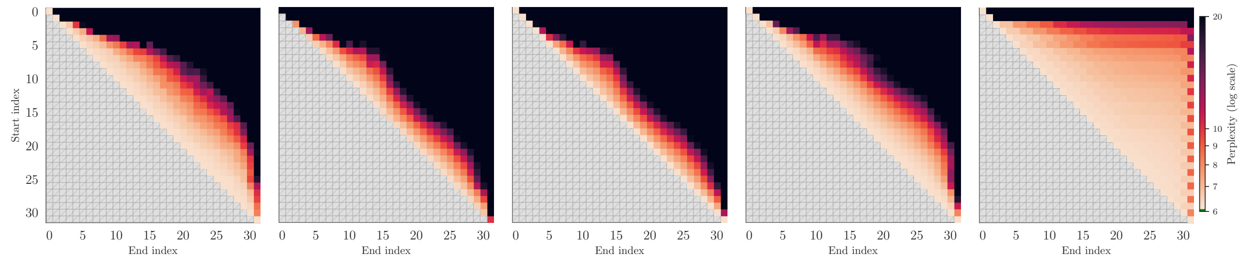


Figure 3: **Diagram of transformations applied in § 3.** Diagrams **(a,b,c,d)** represent shuffling, merging, pruning and parallel respectively.

Parallelism via Computational Graph Optimization. Recent research has investigated architectural layer-level optimization strategies to enhance transformer model inference efficiency. The Staircase Transformer (Cutler et al., 2025) implements parallel layer execution with dynamic recurrent computation based on model requirements. Similarly, the Staggering Transformer (Cai et al., 2024) achieves layer parallelization by connecting layer l_k at time step t to both the $(t-1)$ output of layer l_{k-1} and the t output of layer l_{k-2}. To the best of our knowledge, no research has addressed the fusion of consecutive layers through tensor parallelism.



(a) Shuffling. (b) Pruning. (c) Merging. (d) Parallel. (e) 2-Parallel.

Figure 4: **Changes in perplexity when applying transformations on contiguous stretches of layers.** Each one of the five heatmaps above correspond to a transformation of a group of consecutive layer, where the row index s corresponds to the first layer of the group, and the column index e to the last. The color coding indicates how the perplexity—estimated on a subset of RedPajama (Together Computer, 2023)—is impacted by the corresponding modification of the model. The perplexity for the base Llama 2 7B model is 6.2. In **(a)**, we shuffle—for each forward—the layers from s to e. We can see that many consecutive layers can be shuffled with little impact on the overall perplexity. For instance, shuffling layers 15 to 25—10 layers in total—raises the perplexity only to 9.1. In **(b)**, we prune contiguous stretches of layers. We can see that not many blocks can be removed without starting to significantly degrade the perplexity. In **(c)** we merge contiguous layers. The results with merging are near identical to those for pruning. This reveals there is no advantage in merging layers, most likely a results of averaging matrices not originating from the same initial values. In **(d)** we run contiguous blocks in parallel. Given the success of shuffling, it makes sense that this approach works well. Running blocks 17 to 27 raises the perplexity to 9.3. Finally, in **(e)** we run *pairs of consecutive layers* in parallel. As a result, we can parallelize much longer stretches of layers. For instance, we can apply this transformation from layer 4 to 29 and only increase the perplexity to 9.1. This reduces the depth of the model from 32 to 19. This result makes it possible for us to leverage this parallelism for faster inference as we discuss in § 4.

3 Effective Depth

In this section we investigate the effective depth of pretrained LLMs by applying several transformations and measuring the resulting perplexity degradation. We reveal loose dependencies between intermediary layers. The transformations consist of shuffling, merging, and pruning transformer layers. To avoid the combinatorial explosion resulting from considering all the possible subsets of transformer layers, we instead apply our transformations to all the contiguous stretch of layers. If $L = \{\ell_1, \ldots, \ell_N\}$ are the ordered layers, then we apply our transformations to all the sub-list $\{\ell_i\}_{i=s}^{e}$ with $1 \leq s \leq e \leq N$. Previous works have shown that—at least when considering pruning—the importance of layers is well behaved, with low importance layers close to one another (Men et al., 2024), which justifies considering contiguous stretch of layers only.

Shuffling, pruning and merging blocks. We start by investigating the effect of several transformations on the model's perplexity. First, we experiment with shuffling contiguous stretches of layers (Fig. 3a), re-ordering them according to random permutations. Results, shown in Fig. 4(a), reveal that while shuffling the early and late layers is detrimental, there are large stretches of intermediate blocks that can be shuffled with surprisingly low impact on perplexity. For instance, one can shuffle layers 15 through 24 of Llama 2 7B and only increase perplexity by 2.9. This suggests that many layers may operate at a similar level of abstraction, challenging the classical belief of strictly hierarchical representations. This observed layer decoupling is a key insight. We also experiment with pruning (Fig. 3c) and merging (Fig. 3b) contiguous layers. Pruning, studied in prior works (Gromov et al., 2024; Jung et al., 2019), involves removing layers entirely. Merging involves averaging the weights of consecutive layers. We find that both transformations lead to a more significant perplexity increase compared to shuffling (see Fig. 4b and Fig. 4c). Merging, in particular, offers no advantage over pruning, suggesting that naively combining weights from different layers is ineffective. These initial experiments indicate that while layers are robust to reordering, their individual parameters are crucial.

Running blocks in parallel. The observed layer decoupling suggests that specific transformer operations may be executed independently, providing an opportunity for parallel computation. More precisely, let's consider two sequential transformer layers ℓ_k and ℓ_{k+1}, each comprising attention and Feed-Forward Network (FFN) sub-blocks ($A_k(\cdot)$ and $F_k(\cdot)$, respectively). The standard sequential output $\boldsymbol{y}$ for these layers, given an input $\boldsymbol{x}$, is given by:

$$\begin{aligned}\boldsymbol{y} = \boldsymbol{x} &+ \mathrm{A}_k(\boldsymbol{x}) \\ &+ \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})) \\ &+ \mathrm{A}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}))) \\ &+ \mathrm{F}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})) \qquad (1) \\ &+ \mathrm{A}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})))) \qquad \text{(SEQ)}\end{aligned}$$

The highlighted terms represent the first block's contribution to the second block's processing. Given the observed layer independence, we can hypothesize that these terms have minimal impact, leading to the following approximation:

$$\begin{aligned}\hat{\boldsymbol{y}} = \boldsymbol{x} &+ \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})) \qquad (2) \\ &+ \mathrm{A}_{k+1}(\boldsymbol{x}) + \mathrm{F}_{k+1}(\boldsymbol{x} + \mathrm{A}_{k+1}(\boldsymbol{x})) \qquad \text{(PAR)}\end{aligned}$$

This approximation enables parallel execution of blocks ℓ_k and ℓ_{k+1} through divergent computational paths. We experiment with running contiguous stretches of layers in parallel and show our results in Fig. 4d. We observe results similar to shuffling. Unlike shuffling, this approach allows us to potentially improve the runtime through enhanced parallelism. We show how we can, for instance, run layers 17 to 27 in parallel, only losing 3.1 perplexity points, while reducing the depth of the model from 32 to 23.

Contiguous 2-parallel. Instead of parallelizing long stretches of layers, we experiment with running *pairs of consecutive layers* in parallel. This springs from the assumption that local ordering matters less than global ordering, i.e. shuffling consecutive layers introduces fewer potential issues than shuffling layers separated by larger distances. As an example, if we apply the proposed transformation to layers $\{\ell_{15}, \ell_{16}, \ell_{17}, \ell_{18}, \ell_{19}\}$, it would result in the following process: (1) the two layers $\{\ell_{15}, \ell_{16}\}$ process the input in parallel (according to equation (PAR)), (2) the output is forwarded to layers $\{\ell_{17}, \ell_{18}\}$ which process it in parallel; finally, in (3) their joint output is fed to layer ℓ_{19} which processes it on its own as any normal layer. The effect of such transformation of the compute graph on the perplexity can be seen in Fig. 4e. Remarkably, it is possible to run wide stretches of consecutive pairs of blocks in parallel with only a minor degradation of perplexity. For instance, one can apply this transformation from layer 4 to layer 29 with only a degradation of perplexity of 2.9, while reducing the model depth from 32 to 19. The success of this approach led us to also try running triplets of consecutive layers in parallel, but we found it to perform worse.

4 Efficient Parallelization of Blocks

Naively trying to fuse two attention or MLP sub-blocks does not result in a noticeable improvement of inference speed for large batch sizes and sequence lengths, since in these situations, inference approaches the the compute-bound regime. For this reason we focus our attention on the Tensor Parallel setting, where each module's weights are split over two or more GPUs. Rearranging the computational graph of two contiguous layers like in Fig. 2b effectively reduces the number of inter-GPU synchronizations by half. Now, each divergent path is computed in parallel over multiple accelerators, and only the single intermediate and final results need to be synchronized. While this approach is not numerically equivalent to (PAR), we nonetheless—and quite surprisingly— show that it works well on already trained models, circumventing the need to train from scratch.

LP Multi-Head Attention. Traditional tensor parallelism in MHA distributes attention heads evenly across GPUs (Shoeybi et al., 2020), performing self-attention and output projection locally before gathering

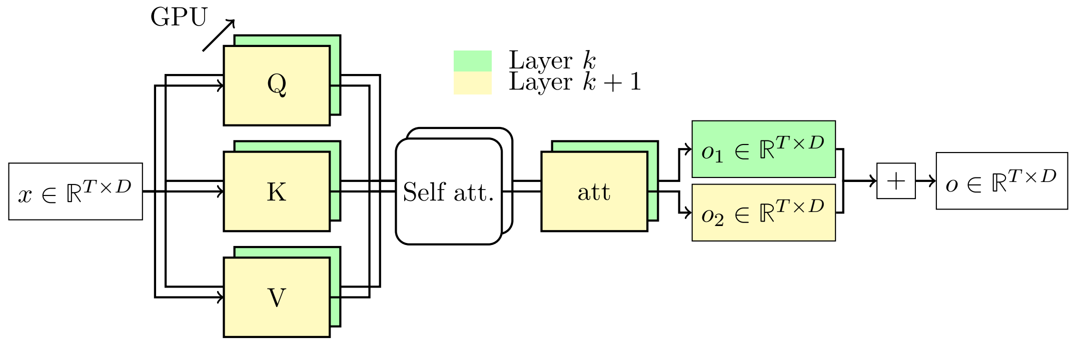


Figure 5: **LP attention implementation**. This diagram shows the implementation of the LP attention from Fig. 2b. The stacked layers represent different GPUs, the colors indicate different layers and the arrows express linear projections. In this case, the number of GPUs and the number of parallelized layers coincides and is two, which is the set-up that we use for all our experiments in this work.

results through an all-reduce summation. Each GPU processes tensors of dimensions $Q, K, V, att \in \mathbb{R}^{T\times \frac{D}{g}}$, where T is sequence length, D is feature dimension, and g is the number of parallel workers. The local output projection produces a low-rank $o_i \in \mathbb{R}^{T\times D}$ for each worker i, and then a final all-reduce operation performs $o = \sum_i^g o_i$, computing the final output.

To implement LP, we increase the depth of the query, key, and value weight matrices ($W_Q, W_K, W_V \in \mathbb{R}^{(g_n\cdot h_d)\times D}$) and widen the output projection ($W_O \in \mathbb{R}^{D\times(n_h\cdot h_d)}$), where n_h represents heads per GPU and h_d is head dimensionality. The reduction operation now will simultaneously compute the full-rank output projections and the sum of all parallel layers (Fig. 5).

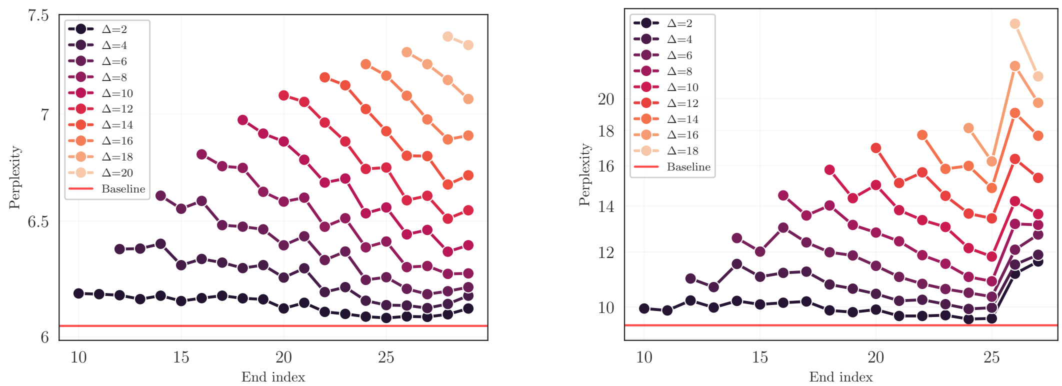
LP FFN. Standard tensor parallelism for single-hidden-layer FFNs splits the first layer's output across devices, generates low-rank outputs from the second layer, and sums them through reduction. To parallelize two FFN layers, we double the first layer's output dimensionality and perform separate output (low-rank) projections for each layer. A single reduction operation then serves the dual purpose of computing full outputs for each layer and combining their results, as shown in Fig. 2(b). In summary, LP for FFN just concatenates the up-projection weights and continues as normal TP, allowing for multiple GPUs to be allocated per parallelized layer.

Handling of the LayerNorms. Since we assume at least one GPU per parallelized layer, we can assign each original LayerNorm to the divergent path that contains the attention and FFN blocks from its original layer. We have also observed that using the same merged LayerNorm with linear interpolation and spherical linear interpolation in each divergent path yields good results. For the sake of simplicity, we conduct all of our experiments using the original LayerNorms on each divergent path.

5 Experiments & Results

In this section, we evaluate Layer Parallelism across three dimensions: inference speed improvements, impact on In-Context Learning performance, and the potential to recover model accuracy through targeted fine-tuning of parallelized layers.

Experimental protocol. For all our experiments, we use a node with x2 A100 SXM4 80Gb GPUs, x4 AMD EPYC 7742 CPUs, and 512Gb of RAM. We test for varying sequence lengths, up to 4096 (Llama's context window), with a batch size of 1 unless indicated otherwise. We consider two models of the Llama family: Llama2 7B, and Llama3.2 3B, as well as two sizes from Qwen3: 4B and 14B. Given a desired effective depth, we replace the required number of normal layers with LP layers. For Llama 2 7B and 3.2 3B, the LP layers are selected based on the configuration that minimized the PPL for a given amount of LP (Fig. 6). For Qwen3, LP is applied until the 4th to last decoder layer. The rest of the layers implement the tensor parallel approach as described in (Shoeybi et al., 2020). For evaluation, we measure the ICL 5-shot



(a) Llama2 7B.

(b) Llama3.2 3B.

Figure 6: **Perplexity when running pairs of consecutive layers in parallel.** Perplexity of Llama2 7B and Llama3.2 3B models on the test set of RedPajama(Together Computer, 2023) when applying Layer Parallelism to Δ consecutive layers. The parallelized interval for each data point is [end index$-\Delta$, end index], where *end index* is the last layer in the LLM to which LP was applied.

accuracies using the `lm-eval` package (Gao et al., 2024). We test the ICL accuracy of the models on several tasks: MMLU (Hendrycks et al., 2021), PiQA (Bisk et al., 2019), ARC Easy, ARC Challenge, Winogrande (Sakaguchi et al., 2021), OpenBookQA (Mihaylov et al., 2018), Hellaswag (Zellers et al., 2019), GSM-8K (Cobbe et al., 2021) and ifeval (Zhou et al., 2023). The perplexity of the models is always evaluated against a subset of the test set of RedPajama (Together Computer, 2023).

Impact of LP on PPL and ICL accuracies. We first examine how perplexity evolves when applying LP across layer sequences of different lengths and depths. Fig. 6 reveals a common optimal sequence end-index minimizing perplexity, found at layers 28 for Llama2 7B and 25 for Llama3.2 3B. Table 1 compares the In-Context Learning performance across models with varying effective depths. Performance declines gradually as LP increases, followed by a sharp drop beyond a certain threshold. Specifically, this occurs after reducing effective depth by 9 layers for Qwen3 14B, by 7 layers for Llama2 7B and Qwen3 4B, and by 5 layers for Llama3.2 3B. These results indicate that larger models are more robust to the computational graph modifications from LP, suggesting that our approach is likely applicable to current commercial-scale LLMs used in major deployments.

It is worth noting that, unlike the other benchmarks, GSM-8K already drops severely in performance when applying low amounts of LP. Recent mechanistic interpretability research shows that LLMs have special circuitry for math operations, localized in a small set of parameters (Stolfo et al., 2023; Yu & Ananiadou, 2024). (Christ et al., 2025) identify math specific parameters, and report a drop in accuracy of 17% when pruning them. We hypothesize that the changes in the computational graph by the use of LP in some of the late layers of the LLM interfere with these fragile and sparse subnetworks, while leaving general language competence largely intact.

Impact on the inference speed. We run an ablation over several configurations and input sequence lengths on Figure 7 to test the speed on three different tasks: KV-Cache pre-filling, autoregressive generation up to the sequence length(with KV-Cache) and 1-token generation with a pre-filled KV-Cache of the corresponding sequence length. Our ablations show that the speed gain is strongly correlated with the reduction of the effective depth of the model. For the effective depths of 25 ($\Delta = 14$) in Llama 2 7B, we observe an average speed-up of 1.29x at the largest sequence length in the 1-token generation task. Likewise, for an effective depth of 23 ($\Delta = 10$) in Llama 3.2 3B, we report a speed-up of 1.22x. For more aggressive parallelism, $\Delta = 18$ and $\Delta = 16$, we report a speed-up of 1.38x and 1.35x , at the expense of a large drop in ICL accuracy.

Fine-tuning for performance recovery. While LP provides speed improvements, associated architectural modifications may degrade model performance. To counteract this, we explored whether fine-tuning could

Table 1: **5-shot In-Context Learning accuracies across standard benchmarks**. Effective Depth shows the minimum number of sequential operations from input to output after applying LP. We use the ablation on Fig 6 to choose the LP configurations that minimized the perplexity. *ifeval was evaluated on 0-shot performance.

Eff. Depth	MMLU	PiQA	Arc E.	Arc C.	WinoG	OBQA	hswag	GSM8K	ifeval*	Avg.
Llama 2 7B										
32 (Base)	0.4583	0.8009	0.8106	0.5196	0.7411	0.4520	0.7821	—	—	0.6521
27 (Ours)	0.4625	0.7933	0.8005	0.5094	0.7348	0.4600	0.7782	—	—	0.6484
26 (Ours)	0.4588	0.7927	0.7976	0.4983	0.7340	0.4460	0.7745	—	—	0.6431
25 (Ours)	0.4532	0.7851	0.7917	0.4949	0.7340	0.4440	0.7673	—	—	0.6386
24 (Ours)	0.4083	0.7845	0.7841	0.4839	0.7190	0.4360	0.7578	—	—	0.6248
23 (Ours)	0.3519	0.7829	0.7677	0.4488	0.6922	0.4240	0.7368	—	—	0.6006
Llama 3.2 3B										
28 (Base)	0.5610	0.7992	0.7807	0.4872	0.7214	0.4520	0.7557	—	—	0.6510
24 (Ours)	0.5508	0.7856	0.7521	0.4753	0.7167	0.4420	0.7384	—	—	0.6373
23 (Ours)	0.5481	0.7748	0.7399	0.4735	0.7119	0.4200	0.7303	—	—	0.6284
22 (Ours)	0.4693	0.7666	0.7264	0.4497	0.6914	0.4180	0.7193	—	—	0.6058
21 (Ours)	0.3890	0.7519	0.6839	0.4061	0.6638	0.4020	0.6847	—	—	0.5690
20 (Ours)	0.3107	0.7416	0.6481	0.3652	0.6227	0.3620	0.6407	—	—	0.5273
Llama 2 7B (Chat)										
32 (Base)	0.4727	0.7769	0.7963	0.4906	0.7206	0.3300	0.5887	0.2297	0.4365	0.5380
27 (Ours)	0.4746	0.7709	0.7803	0.4881	0.7182	0.3380	0.5839	0.2221	0.4736	0.5389
26 (Ours)	0.4724	0.7666	0.7748	0.4701	0.7151	0.3400	0.5779	0.1903	0.4856	0.5325
25 (Ours)	0.4767	0.7633	0.7719	0.4650	0.7009	0.3320	0.5724	0.1478	0.4796	0.5233
24 (Ours)	0.4547	0.7655	0.7567	0.4369	0.6788	0.3040	0.5582	0.0963	0.4149	0.4962
23 (Ours)	0.4271	0.7524	0.7428	0.4113	0.6654	0.3080	0.5406	0.0697	0.3765	0.4771
Llama 3.2 3B (Instruct)										
28 (Base)	0.5956	0.7709	0.7929	0.4650	0.7024	0.3080	0.5253	0.6475	0.6715	0.6088
24 (Ours)	0.5914	0.7622	0.7660	0.4488	0.7009	0.3000	0.5131	0.4587	0.6031	0.5716
23 (Ours)	0.5905	0.7546	0.7618	0.4471	0.6898	0.2780	0.5116	0.3571	0.5995	0.5544
22 (Ours)	0.5612	0.7530	0.7504	0.4573	0.6677	0.2880	0.5088	0.1001	0.5564	0.5159
21 (Ours)	0.5072	0.7470	0.7256	0.4019	0.6409	0.2840	0.4947	0.0311	0.4904	0.4803
20 (Ours)	0.4070	0.7410	0.7075	0.3942	0.6275	0.2900	0.4755	0.0311	0.4448	0.4576
Qwen3 4B (Instruct)										
36 (Base)	0.7016	0.7644	0.8476	0.5879	0.6630	0.3720	0.5277	0.8499	0.4209	0.6372
31 (Ours)	0.6887	0.7459	0.8182	0.5367	0.6598	0.3520	0.5010	0.5375	0.3549	0.5772
30 (Ours)	0.6749	0.7497	0.8184	0.5239	0.6511	0.3340	0.4895	0.3677	0.3813	0.5545
29 (Ours)	0.6356	0.7470	0.7988	0.5043	0.6322	0.3300	0.4725	0.1797	0.3657	0.5184
28 (Ours)	0.5395	0.7432	0.7929	0.4898	0.6275	0.3000	0.4524	0.1266	0.3465	0.4909
27 (Ours)	0.4409	0.7296	0.7668	0.4454	0.5967	0.2960	0.4322	0.0356	0.2782	0.4468
26 (Ours)	0.3795	0.7062	0.7231	0.4002	0.5675	0.2840	0.4064	0.0318	0.3165	0.4239
Qwen3 14B (Instruct)										
40 (Base)	0.7883	0.8090	0.8775	0.6621	0.7451	0.4040	0.6133	0.8226	0.4652	0.6875
35 (Ours)	0.7792	0.7982	0.8695	0.6468	0.7356	0.3880	0.5825	0.6164	0.4376	0.6504
34 (Ours)	0.7742	0.7889	0.8561	0.6271	0.7388	0.3780	0.5792	0.7536	0.4700	0.6629
33 (Ours)	0.7588	0.7905	0.8561	0.6058	0.7269	0.3740	0.5693	0.6983	0.4376	0.6464
32 (Ours)	0.7370	0.7889	0.8481	0.5973	0.7103	0.3680	0.5575	0.6467	0.4017	0.6284
31 (Ours)	0.7156	0.7856	0.8552	0.5998	0.6859	0.3640	0.5412	0.4822	0.3801	0.6011
30 (Ours)	0.6543	0.7758	0.8384	0.5589	0.6732	0.3460	0.5197	0.2343	0.3657	0.5518
29 (Ours)	0.5954	0.7688	0.8215	0.5256	0.6575	0.3260	0.4945	0.1585	0.3381	0.5207

Table 2: Benchmark accuracy restoration through fine-tuning on Qwen3 4B with an effective depth of 27.

Fine-tuning Steps	MMLU	Arc C.	GSM-8K (%)
0 (Ours)	0.4409	0.4454	0.0356
1024 (Ours)	0.6063	0.4932	0.4011
4096 (Ours)	0.6196	0.5102	0.4829
8192 (Ours)	0.6134	0.5162	0.4943
Qwen3 4B (Base)	0.7016	0.5879	0.8499

effectively restore the original model's capabilities. Specifically, we applied LP to layers 14–32 of Qwen3 4B, subsequently fine-tuning only these modified layers on randomly selected samples from the RedPajama training dataset (Together Computer, 2023). Employing a batch size of 32, a linear learning rate schedule

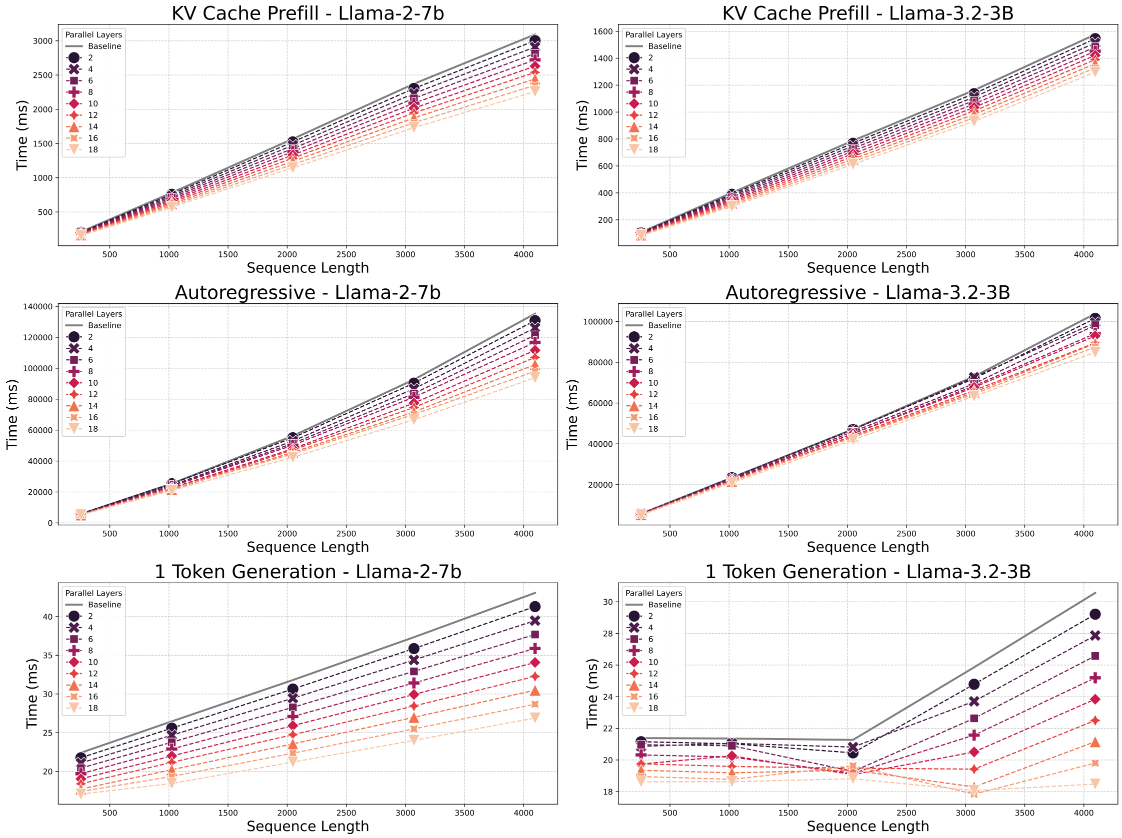


Figure 7: **Wall clock time to complete the following inference tasks**: KV Cache pre-filling, autoregressive generation, and single token generation with a pre-filled KV Cache. Δ indicates how many layers have been merged using LP (e.g. a Δ of 4 indicates that 2 groups of 2 layers have been converted to 2 effective layers). The gains in inference speed are directly proportional to the grade of LP. The 1-token generation task for Llama 3.2 3B does not saturate the GPU compute until a sequence length of 2048. Even in this regime, LP benefits from considerable speed-ups.

starting at $1e{-}4$, and the AdamW optimizer (Loshchilov & Hutter, 2017), we observed significant restoration of the model's performance. Notably, GSM-8K accuracy improved from near-zero levels to more than half of the original model's accuracy. It is possible that additional fine-tuning, or smarter tuning strategies, could yield further improvement, but resource constraints limited the scope of our experiments.

6 Limitations

The effectiveness of our approach exhibits notable variations across model scales. Smaller models show reduced benefits, likely due to their less sparse activation patterns and more tightly coupled layer dependencies. This degradation becomes more pronounced as the LP sequence length increases, suggesting a practical upper limit to the number of layer pairs that can be effectively parallelized.

Regarding the fine-tuning, while some performance loss can be mitigated, we were unable to fully recover the baseline model's performance levels. This suggests fundamental trade-offs between computational efficiency and model capability that cannot be entirely eliminated through optimization, or that more involved fine-tuning strategies might be required.

Moreover, determining the 'true' effective depth—the optimal configuration of parallel layer pairs—remains an open challenge as there is no theoretical framework for predicting the optimal grouping strategy.

These limitations highlight important directions for future research, particularly in developing more robust methods for determining optimal layer groupings and investigating the interplay between our approach and other efficiency-oriented techniques.

7 Conclusion

In this work, we presented Layer Parallelism, a novel approach that exploits independence patterns between transformer layers to optimize LLM inference. By restructuring the computational graph to enable parallel execution of consecutive layer pairs through tensor parallelism, we achieved substantial speed improvements without model retraining. Our method reduced the effective depth of Llama 2 7B by 21% while maintaining 98% of the original performance (without fine-tuning), yielding up to a 1.29x improvement in inference speed for single-token generation with long sequences. Moreover, we show that we can recover some of the lost accuracy through naive fine-tuning.

These results challenge the conventional view that transformer layers must process information strictly sequentially, suggesting instead that certain layers can operate independently without significant performance loss. From a practical standpoint, LP offers a straightforward approach to improve inference efficiency in production environments. Future work could focus on developing theoretical frameworks to predict optimal layer groupings, investigating interactions with other efficiency techniques such as quantization, and understanding the fundamental principles behind layer independence. Despite its limitations, LP represents a practical advancement in making LLM deployment more efficient and economically viable.

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Bryan R. Christ, Zack Gottesman, Jonathan Kropko, and Thomas Hartvigsen. Math neurosurgery: Isolating language models' math reasoning abilities using only forward passes, 2025. URL https://arxiv.org/abs/2410.16930.

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. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Dylan Cutler, Arun Kandoor, Nishanth Dikkala, Nikunj Saunshi, Xin Wang, and Rina Panigrahy. Stagformer: Time staggering transformer decoding for runninglayers in parallel, 2025. URL https://arxiv.org/abs/2501.15665.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, 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, Danny Wyatt, 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, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, 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, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2024. URL https://arxiv.org/abs/2403.17887.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. URL https://arxiv.org/abs/1510.00149.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pp. 10–14. IEEE, 2014.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

Hyun-Joo Jung, Jaedeok Kim, and Yoonsuck Choe. How compact?: Assessing compactness of representations through layer-wise pruning, 2019. URL https://arxiv.org/abs/1901.02757.

Vedang Lad, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference?, 2024. URL https://arxiv.org/abs/2406.19384.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Neural Information Processing Systems*, 1989. URL https://api.semanticscholar.org/CorpusID:7785881.

Zonghang Li, Wenjiao Feng, Mohsen Guizani, and Hongfang Yu. Tpi-llm: Serving 70b-scale llms efficiently on low-resource edge devices, 2024. URL https://arxiv.org/abs/2410.00531.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm, 2021. URL https://arxiv.org/abs/2104.04473.

OpenAI. GPT-4 Technical Report, March 2023. URL http://arxiv.org/abs/2303.08774. arXiv:2303.08774 [cs].

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert, 2019. URL https://arxiv.org/abs/1909.05840.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Aditi Singh, Nirmal Prakashbhai Patel, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. A survey of sustainability in large language models: Applications, economics, and challenges, 2025. URL https://arxiv.org/abs/2412.04782.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks, 2015. URL https://arxiv.org/abs/1505.00387.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. *arXiv preprint arXiv:2305.15054*, 2023.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. URL https://arxiv.org/abs/1409.4842.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks, 2017. URL https://arxiv.org/abs/1709.01686.

Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

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, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, pp. 550–558, 2016.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *HPCA*, pp. 97–110. IEEE, 2021.

Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, James Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. Sustainable ai: Environmental implications, challenges and opportunities, 2022. URL https://arxiv.org/abs/2111.00364.

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, Qiyang Zhang, Zhenyan Lu, Li Zhang, Shangguang Wang, Yuanchun Li, Yunxin Liu, Xin Jin, and Xuanzhe Liu. A survey of resource-efficient llm and multimodal foundation models, 2024. URL https://arxiv.org/abs/2401.08092.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zeping Yu and Sophia Ananiadou. Interpreting arithmetic mechanism in large language models through comparative neuron analysis. *arXiv preprint arXiv:2409.14144*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. Sparch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 261–274. IEEE, 2020.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit, 2020. URL https://arxiv.org/abs/2006.04152.

A Ablation: Tokens per second

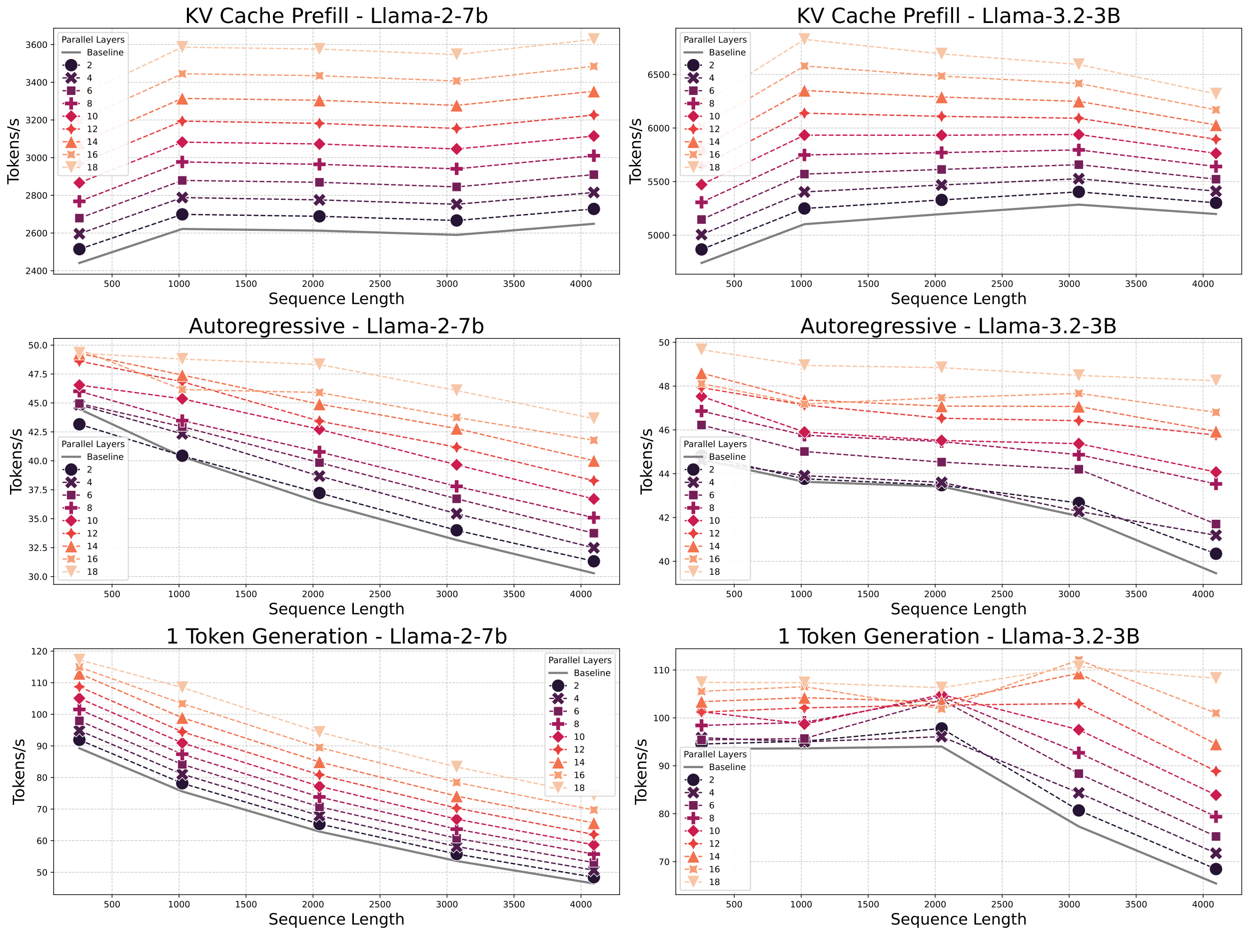


Figure 8: Tokens per second when completing the following inference tasks: KV Cache pre-filling for a given sequence length, autoregressive generation up to the indicated sequence length, and single token generation with a pre-filled KV Cache of the indicated sequence length. The baseline is the original model with all layers making use of Tensor Parallelism. The Parallel Layers number (Δ) indicates how many layers have been merged using Layer Parallelism (e.g. a Δ of 4 indicates that 2 groups of 2 layers have been converted to 2 effective layers). The number of tokens is computed as the sum of the input tokens and the output tokens for each forward pass.

B Generalization to multiple GPUs

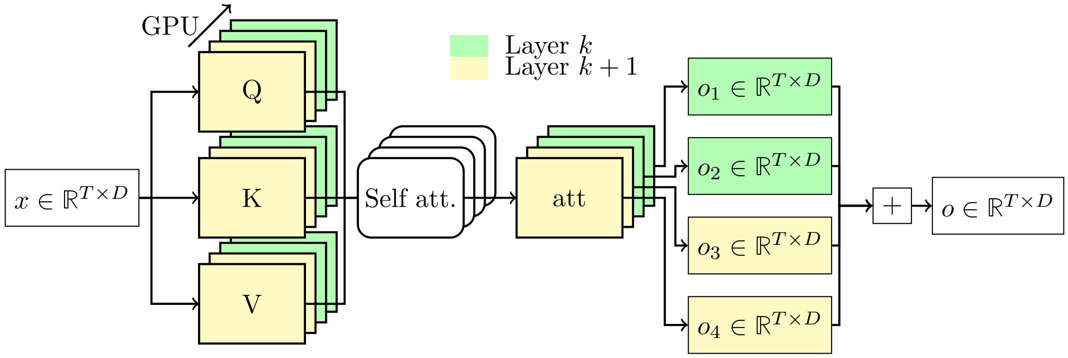
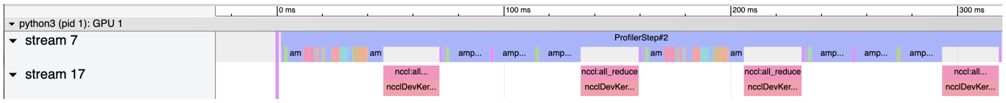


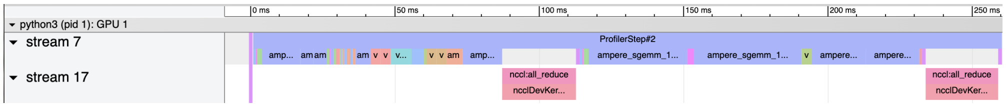
Figure 9: Layer Parallelism in the case of parallelizing two layers over four accelerators. The stacked layers represent the tensor parallelism, and the colors indicate the processing of different previously contiguous layers. $Q, K, V, \text{att} \in \mathbb{R}^{T\times \frac{2D}{g}}$, where D is the feature dimension and g is the total number of accelerators.

Layer Parallelism allows one to allocate $N \geq 1$ accelerators for each layer. The implementation remains the same, but now each layer is parallelized using tensor parallelism over its assigned accelerators. Note that both reduction operations (tensor parallel and layer parallel) are nicely executed with a single all-reduce call.

C Acceleration source



(a) Flame chart of running two standard Tensor-Parallel LLama decoder layers.



(b) Flame chart of running two Llama 3 decoder layers with our Layer Parallelism approach.

Figure 10: Comparison of Flame Graphs when running two consecutive Llama 3.2 3B decoder layers with vanilla tensor parallelism (Fig. 10a), and our Layer Parallelism approach (Fig. 10b). Note that the time axis scale is different between both graphs. These results were obtained on a workstation using x2 RTX 4090s.

Figure 10 illustrates flame graphs comparing two consecutive Llama 3.2 3B decoder layers using vanilla tensor parallelism and our LP approach. The profiling data summarized in Table 3 reveals that the primary source of acceleration in our LP method stems from reducing the total number of `all-reduce` synchronization operations across GPUs. Specifically, the vanilla tensor parallel approach performs synchronization at every decoder layer, resulting in higher cumulative synchronization overhead. In contrast, our Layer Parallelism implementation runs pairs of layers simultaneously, effectively halving the number of synchronization points. This reduction in synchronization leads to a significant drop in synchronization time from 100.8ms to 50.7ms, directly contributing to the observed improvement in inference speed.

Additionally, Layer Parallelism enables fusion of certain computation kernels—particularly attention and MLP operations across parallelized layers—which further marginally reduces the computation time from 217ms to 208.7ms. Although these computational gains are modest compared to the savings achieved through fewer synchronization operations, kernel fusion further optimizes hardware utilization and enhances overall throughput.

Table 3: Profiling results comparing vanilla Tensor Parallel and Layer Parallel implementations on two consecutive Llama 3.2 3B decoder layers.

Approach	**Total Time (ms)**	**Sync Time (ms)**	**Computation Time (ms)**
Tensor Parallel	317.8	100.8	217.0
Layer Parallel (Ours)	259.4 (x1.23)	50.7 (x1.99)	208.7 (x1.04)