Leveraging the true depth of LLMs

Anonymous authors
Paper under double-blind review

Abstract

Large Language Models demonstrate remarkable capabilities at the cost of high compute requirements. While recent research has shown that intermediate layers can be removed or have their order shuffled without impacting performance significantly, these findings have not been employed to reduce the computational cost of inference. We investigate several potential ways to reduce the depth of pre-trained LLMs without significantly affecting performance. Leveraging our insights, we present a novel approach that exploits this decoupling between layers by grouping some of them into pairs that can be evaluated in parallel. This modification of the computational graph—through better parallelism—results in an average improvement of around 1.20x on the number of tokens generated per second, *without re-training nor fine-tuning*, while retaining 95%-99% of the original accuracy. Empirical evaluation demonstrates that this approach significantly improves serving efficiency while maintaining model performance, offering a practical improvement for large-scale LLM deployment.

1 Introduction

The exponential growth of Large Language Models (LLMs) presents significant computational challenges for commercial deployment, with critical implications for costs, performance, and environmental impact (Singh et al., 2025; Xu et al., 2024; Wu et al., 2022). High-traffic LLM applications can incur monthly cloud computing costs in the millions, emphasizing the importance of optimization.

Modern LLMs utilize deep architectures with hundreds of transformer layers (OpenAI, 2023; et. al., 2024), featuring attention and feed-forward blocks connected by residual connections similar to ResNets (He et al., 2015). Research has shown that network depth may be partially redundant, allowing for layer reorganization without significant performance impact (Veit et al., 2016). Recent studies have extended these findings to transformer architectures (Lad et al., 2024), demonstrating resilience to layer modifications.

Our research investigates depth reduction in LLMs through various interventions including shuffling, pruning, and merging layers. The ability to reorder blocks enables parallel processing strategies, allowing multiple block pairs to be executed simultaneously while maintaining acceptable performance on perplexity and In-Context Learning benchmarks. We propose Layer Parallelism (LP), a novel method that improves inference speed with minimal performance degradation, which can be further recovered through targeted fine-tuning.

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 loosing significant performance. Our approach can be applied to existing Transformer models and *does not require re-training.*
- We exploit this parallelization of the computational graph to run the models around 1.20x faster using multiple GPUs without loosing much performance and ICL capabilities.

- We show that by fine-tuning a parallelized model we can recover some of the lost performance, while retaining the previously obtained speed-up.

The source code will be made publicly available upon publication of this paper.

2 Related work

The effective depth of Deep Networks. Theoretically, any feed-forward network with at least one hidden layer can model any function, given enough width (Pinkus, 1999). In practice, it is easier to achieve high expressivity by increasing the model's depth. But naively increasing the depth can make things more difficult for the optimizer, 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, in Inception (Szegedy et al., 2014), the researchers investigated ways 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.

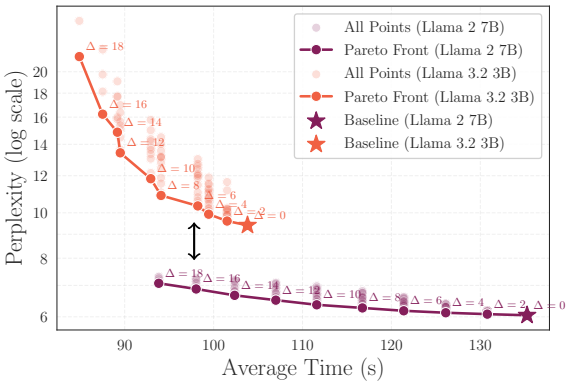
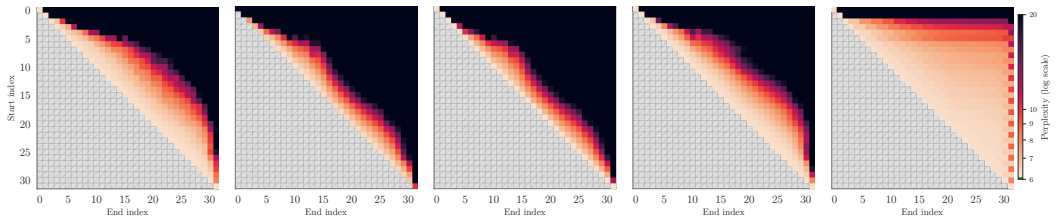


Figure 1: **Trade-off between perplexity and inference time of Llama 2 7B and Llama 3.2 3B with our novel Layer Parallelism**. Δ indicates the number of consecutive layers that we parallelize in pairs. The average time is the wall clock time to generate 4096 tokens (with KV-Cache) on x2 A100 SXM4 80Gb. The perplexity is measured on RedPajama Together Computer (2023). As shown by the gap between the two Pareto fronts (black arrow), applying our approach to Llama 2 7b results in both faster generation speeds and better performance than the vanilla Llama 3.2 3B.

Efficient inference of LLMs. Several complementary approaches exist for enhancing the computational efficiency of large-scale models, primarily through pruning and sparsity, quantization, and parallelism. PruningLeCun 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 contribution 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). In contrast, quantization encompasses the transformation of floating-point numerical representations (predominantly FP32) into reduced-precision integer formats, such as INT8 or INT4 Han et al. (2016); Jacob et al. (2018). When implemented on hardware accelerators, these lower-precision representations facilitate superior memory bandwidth utilization, 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 devices, 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).

Parallelism via Computational Graph Optimization. Recent research has investigated architectural layer-level optimization strategies to enhance transformer model inference



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

Figure 3: **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.

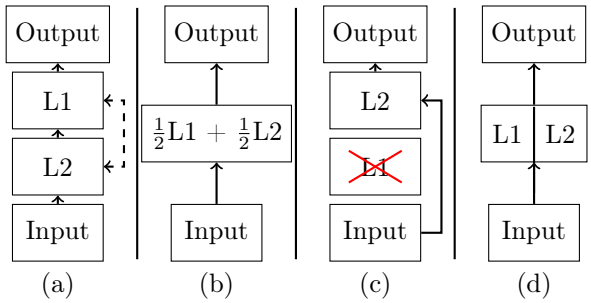


Figure 4: **Diagram of transformations applied in § 3.** Diagrams **(a,b,c,d)** represent respectively shuffling, merging, pruning and parallel.

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.

3 Effective Depth

In this section, we investigate the effective depth of LLMs. By applying several transformations to a pre-trained Transformer LLM, and measuring the resulting perplexity degradation, we reveal the 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 blocks. We start by shuffling contiguous stretches of layers, re-ordering the layers according to random permutations. Results are shown in Fig. 3.(a). While shuffling the early and two last layers significantly raises the perplexity, there are large stretches of blocks which can be shuffled with surprisingly low impact on the perplexity. For instance, one can shuffle the layers $\{\ell_i\}_{15 \leq i < 25}$ and only have an increase in perplexity of 2.9. This goes against the classical belief that models build deeply hierarchical representations, where features in previous layers are leveraged to build more complex features in later layers. We interpret this as multiple layers working at the same level of abstraction. Using these insights, we define the *effective depth* of an LLM as the shortest depth required to efficiently leverage existing latent representations without a significant loss of performance. This reveals an important level of layer decoupling within the model.

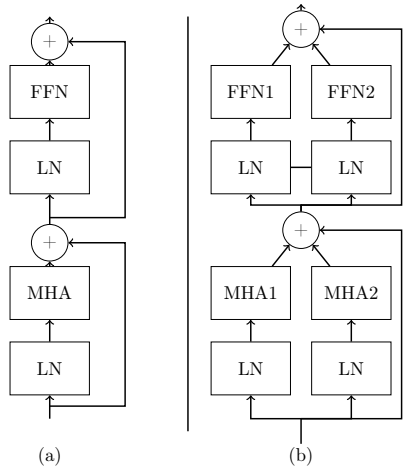


Figure 2: **Comparing a normal transformer block with our layer parallel implementation.** In **(a)** we show a normal Transformer Layer. In **(b)** we illustrate our layer parallelization, where each column runs separately on one or multiple GPUs. The residuals are gathered and synchronized through a reduce operation across all GPUs. The pre-attention LayerNorms' weights are copied from the original blocks, while the post-attention LayerNorm's weights are averaged and shared in both layers.

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 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:

$$\begin{aligned}\boldsymbol{y} = \boldsymbol{x} &+ \mathrm{A}_k(\boldsymbol{x}) \\ &+ \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_1(\boldsymbol{x})) \\ &+ \mathrm{A}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}))) \\ &+ \mathrm{F}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})) \qquad (1) \\ &+ \mathrm{A}_{k+1}(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x}) + \mathrm{F}_k(\boldsymbol{x} + \mathrm{A}_k(\boldsymbol{x})))) \qquad (\text{SEQ})\end{aligned}$$

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. 3d. We observe results similar to shuffling. Unlike shuffling, this approach allows us to potentially improve the runtime through enhanced parallelism.

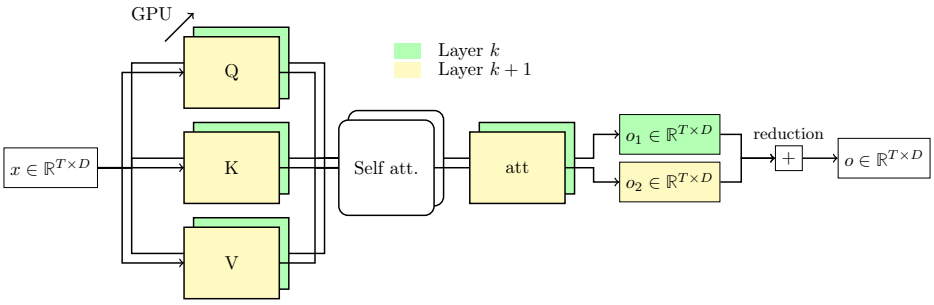


Figure 5: **Layer Parallel implementation of self-attention**. In this diagram, the stacked layers represent different GPUs, the colors indicate the intermediate tensors of different layers and the arrows express linear projections. In this case, the number of GPUs and the number of parallelized layers coincides and is 2, which is the set-up that we use for all our experiments in this work. We see how tensors for the two layers are distributed across the two GPUs. We also notice the reduction step happening at the end, summing the outputs from each GPUs, which differs from the regular computational graph obtained when running layers in parallel as in equation (PAR).

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 is less risky than shuffling layers further apart. 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. 3e. 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 less well.

Exploring other transformations. We also experiment with pruning (Fig. 3b) and merging (Fig. 3c). Pruning has already been studied in several prior works (Gromov et al., 2024; Jung et al., 2019). While reducing the depth of the model by one layer costs more perplexity when pruning compared to running two blocks in parallel, pruning reduces the number of parameters of the model, which directly translates into higher throughput and memory efficiency.

4 Efficient Parallelization of Blocks

Naive block parallelization. Pure parallelization of two transformer blocks would consist in implementing equation (PAR). To optimize our parallel implementation, we need to leverage efficient GPU kernels by e.g. concatenating together matrices from different blocks. As an example, let's consider parallelizing $\boldsymbol{y}_1 = W_1\boldsymbol{x}$ and $\boldsymbol{y}_2 = W_2\boldsymbol{x}$, $\boldsymbol{x} \in \mathbb{R}^{d_x}, \boldsymbol{y} \in \mathbb{R}^{d_y}, W_1, W_2 \in \mathbb{R}^{d_y \times d_x}$. We can concatenate $W = [W_1, W_2] \in \mathbb{R}^{2*d_y \times d_x}$ and get $[\boldsymbol{y}_1, \boldsymbol{y}_2] = W\boldsymbol{x}$. However, this would not work if, instead of a shared input $\boldsymbol{x}$, we had two separate inputs $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. This is precisely the situation we are in when we apply different layer norms to the input of the two attention blocks. The Feed-Forward Networks (FFN) also have different inputs, in addition to having different layer norms. Those considerations directed us to modify the computational graph of the parallel processing of blocks. While we differ from

equation (PAR), we show that our approach nonetheless—and quite surprisingly—works well on already trained models, circumventing the need to train from scratch.

The hardware limits of parallelisms. Another difficulty we faced when parallelizing transformer blocks is the saturation of GPU resources. LLM inference typically saturates GPU resources due to its intensive compute and memory bandwidth requirements. When all the GPU cores are already being utilized by one matrix multiplication, running another matrix multiplication in parallel can be as slow as running them sequentially. This is the case with large transformer models, as even processing a single sequence can fully utilize a GPU's capabilities. As such, simply attempting to run two blocks in parallel would result in sequential execution, as the scheduler would allocate operations from both blocks to the same job stream. To achieve true parallel execution of layers, we decided to leverage Tensor Parallelism by distributing layer weights across multiple GPUs.

We extend the tensor parallelism scheme introduced in Megatron Shoeybi et al. (2020) to incorporate our novel Layer Parallelism. Below, we detail our approach for each major component of the transformer block, initially setting aside Layer Normalization considerations and tackling the Multi-Head Attentions (MHA) and the FFN. Our approach is also illustrated in Fig. 2. It is important to note that the resulting computational graph is not numerically equivalent to the original architecture. This numerical discrepancy stems from the positioning of the pre-normalization operations that precede each transformer sub-block.

Layer Parallel Multi-Head Attention. Traditional tensor parallelism in MHA distributes attention heads evenly across GPUs, performing self-attention and output projection locally before gathering results through worker 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 $o_i \in \mathbb{R}^{T \times D}$ for each worker i, with rank $\frac{D}{g}$ until the gather operation restores full rank. To implement Layer Parallelism, 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(b)).

Layer Parallel Feed Forward Network. Standard tensor parallelism for single-hidden-layer FFNs splits the first layer's output across devices, generates partial outputs from the second layer, and combines them through reduction. To parallelize two FFN layers, we double the first layer's output dimensionality and perform separate output 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, Layer Parallelism for FFN just concatenates the up-projection weights and continues as normal TP, allowing for multiple GPUs to be allocated per parallelized layer.

Handling Layer Normalization. Layer Normalization presents unique challenges since these layers were trained on specific input distributions. For MHA pre-normalization, we apply separate normalization on each device. For FFN pre-normalization, we found that linearly interpolating the weights of both FFN pre-norms yielded better perplexity than maintaining separate normalizations. This improvement may stem from the fact that, given we reduce the MHA outputs over the parallelized layers, interpolation effectively combines the expected input distributions of both layers.

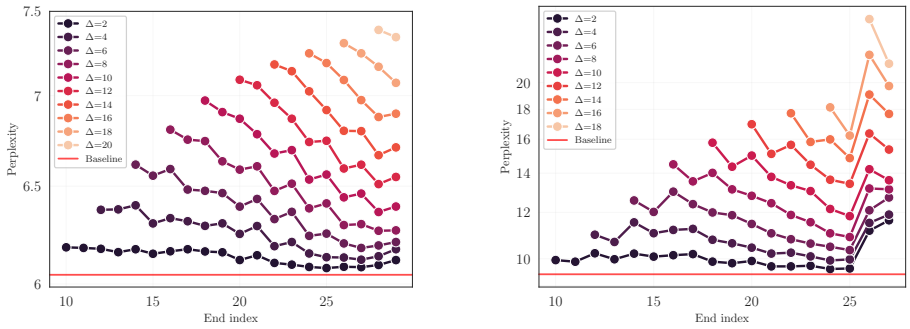
5 Experiments & Results

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 two A100 SXM4 80Gb GPUs, four 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

Table 1: **5-shot In-Context Learning accuracies across standard benchmarks**. Effective Depth shows the minimum number of sequential operations from input to output. Parallel Layers (PL) indicates the range of consecutive layers where pairs were processed in with Layer Parallelism.

Eff. Depth	PL	MMLU	PiQA	Arc E.	Arc C.	WinoG	OBQA	hellaswag
Llama 2 7B								
32 (Baseline)	-	0.4583	0.8009	0.8106	0.5196	0.7411	0.4520	0.7821
27 (Ours)	18-28	0.4625	0.7933	0.8005	0.5094	0.7348	0.4600	0.7782
26 (Ours)	16-28	0.4588	0.7927	0.7976	0.4983	0.7340	0.4460	0.7745
25 (Ours)	14-28	0.4532	0.7851	0.7917	0.4949	0.7340	0.4440	0.7673
24 (Ours)	12-28	0.4083	0.7845	0.7841	0.4839	0.7190	0.4360	0.7578
23 (Ours)	10-28	0.3519	0.7829	0.7677	0.4488	0.6922	0.4240	0.7368
Llama 3.2 3B								
28 (Baseline)	-	0.5610	0.7992	0.7807	0.4872	0.7214	0.4520	0.7557
24 (Ours)	17-25	0.5508	0.7856	0.7521	0.4753	0.7167	0.4420	0.7384
23 (Ours)	15-25	0.5481	0.7748	0.7399	0.4735	0.7119	0.4200	0.7303
22 (Ours)	13-25	0.4693	0.7666	0.7264	0.4497	0.6914	0.4180	0.7193
21 (Ours)	11-25	0.3890	0.7519	0.6839	0.4061	0.6638	0.4020	0.6847
20 (Ours)	9-25	0.3107	0.7416	0.6481	0.3652	0.6227	0.3620	0.6407



(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 RedPajamaTogether Computer (2023) when applying Layer Parallelism to Δ consecutive layers. The parallelized interval for each data point is $[\text{end index} - \Delta, \text{end index}[$.

otherwise. We consider two models of the Llama family: Llama2 7B, and Llama3.2 3B. We always apply Layer Parallelism of 2 (one layer to each GPU) on the merged sequential layers and apply Tensor Parallelism as described in (Shoeybi et al., 2020) for the rest. The baselines are fully Tensor Parallel Llama models. For evaluation, we measure the ICL 5-shot 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 E.), ARC Challenge (Arc C.), Winogrande (WinoG) (Sakaguchi et al., 2021), OpenBookQA (OBQA) (Mihaylov et al., 2018) and Hellaswag (Zellers et al., 2019). The perplexity (PPL) of the models is always evaluated against a subset of the test set of RedPajama (Together Computer, 2023).

Impact of layer-parallelism on PPL and ICL accuracies. We begin by exploring the evolution of the perplexity when applying layer parallelism to stretches of layers of varying lengths, and starting at different depths. Results in Fig. 6 show how both models—Llama2 7B and Llama3.2 3B—exhibit a common sequence ending index for which the perplexity is minimized, which is 28 and 25 for Llama2 7B and Llama3.2 3B, respectively. Taking this into consideration, in Table 1 we evaluate the In-Context Learning capabilities of models of different effective depths, for which parallelized sequences end at those indices. For Llama 2 7B we observe that applying Layer Parallelism to a sequence greater than 14 layers results in a steep loss of In-Context Learning capabilities in the more challenging benchmarks, like

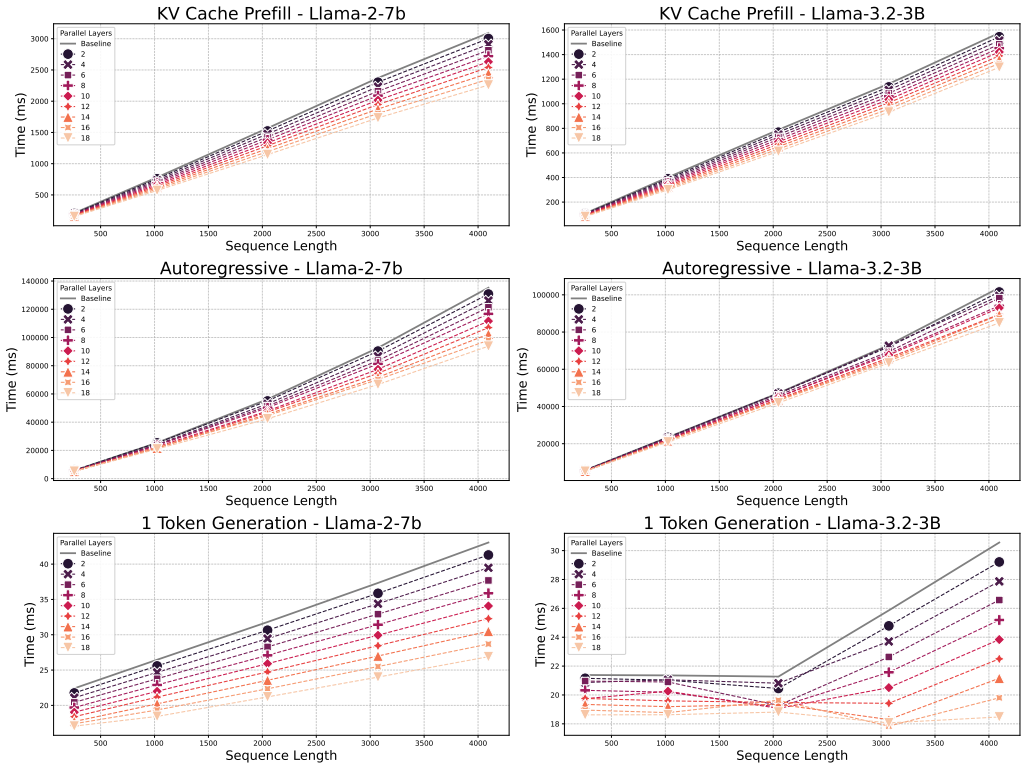


Figure 7: **Wall clock time to complete the following inference tasks**: KV Cache pre-filling for a given sequence length, autoregressive generation up to the indicated sequence length, and single token generation with a pre-filled KV Cache of the indicated sequence length. The baseline is the original 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 gains in inference speed are directly proportional to the grade of Layer Parallelism. 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, Layer Parallelism benefits from considerable speed-ups.

MMLU. Likewise, parallelizing above 10 layers of Llama 3.2 3B sees an even more rapid decrease in performance on difficult benchmarks. The effective depth of both models at the parallel configurations before those sudden drops in performance is 25 and 23, a reduction of 21% and 18% of their original depths, respectively.

Impact on the inference speed. We run an ablation over several configurations and input sequence lengths on Figure 9 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 directly proportional to 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 Layer Parallelism offers significant speed improvements, the architectural modifications can impact model performance. To address this, we investigated whether fine-tuning could recover the original model's capabilities. Using Llama 3.2 3B with Layer Parallelism applied to layers 13-25 ($\Delta = 12$), we fine-tuned only the parallelized layers on random samples from RedPajama's training set Together

Computer (2023). With a batch size of 2 and a learning rate of $1e-4$, we observed substantial recovery of model performance. As shown in Table 2, fine-tuning for 8,192 steps improved MMLU accuracy from 83.6% to 94.4% of the baseline performance, demonstrating that much of the model's original capability can be recovered while maintaining the speed benefits of Layer Parallelism.

6 Limitations

While our approach demonstrates significant improvements in inference efficiency, several important limitations should be considered.

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. Even in successful cases, we observe a consistent, albeit small, performance degradation compared to the baseline. This degradation becomes more pronounced as model depth increases, suggesting a practical upper limit to the number of layer pairs that can be effectively parallelized.

Table 2: **Recovery of MMLU accuracy through fine-tuning on Llama 3.2 3B with Layer Parallelism applied to layers 13-25.** Relative MMLU shows performance as a percentage of the baseline model's accuracy. The recovered MMLU saturates after fine-tuning for 8k steps.

Fine-tuning Steps	MMLU	Rel. MMLU (%)
0 (Baseline)	0.5610	100
0 (13-25 LP)	0.4693	83.6
4096 (Ours)	0.4979	88.8
8192 (Ours)	0.5295	94.4
16384 (Ours)	0.5222	93.1
32736 (Ours)	0.5266	93.8

Finetuning. While some performance loss can be mitigated through fine-tuning, 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.

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 strong performance, yielding up to a 1.29x improvement in inference speed for single-token generation with long sequences. Similar benefits were observed with Llama 3.2 3B, achieving an 18% reduction in effective depth with up to a 1.22x speed-up. Moreover, we show that we can recover 10.8% of ICL accuracy on MMLU by fine-tuning the parallelized models using few resources.

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

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Cutler, D., Kandoor, A., Dikkala, N., Saunshi, N., Wang, X., and Panigrahy, R. Stagformer: Time staggering transformer decoding for runninglayers in parallel, 2025. URL https://arxiv.org/abs/2501.15665.

et. al., A. G. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Gholami, A., Yao, Z., Kim, S., Hooper, C., Mahoney, M. W., and Keutzer, K. Ai and memory wall. *IEEE Micro*, 2024.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers, 2024. URL https://arxiv.org/abs/2403.17887.

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

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Horowitz, M. 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.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. 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.

Jung, H.-J., Kim, J., and Choe, Y. How compact?: Assessing compactness of representations through layer-wise pruning, 2019. URL https://arxiv.org/abs/1901.02757.

Lad, V., Gurnee, W., and Tegmark, M. The remarkable robustness of llms: Stages of inference?, 2024. URL https://arxiv.org/abs/2406.19384.

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

Li, Z., Feng, W., Guizani, M., and Yu, H. Tpi-llm: Serving 70b-scale llms efficiently on low-resource edge devices, 2024. URL https://arxiv.org/abs/2410.00531.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL https://arxiv.org/abs/2403.03853.

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

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. 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].

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

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Singh, A., Patel, N. P., Ehtesham, A., Kumar, S., and Khoei, T. T. A survey of sustainability in large language models: Applications, economics, and challenges, 2025. URL https://arxiv.org/abs/2412.04782.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks, 2015. URL https://arxiv.org/abs/1505.00387.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions, 2014. URL https://arxiv.org/abs/1409.4842.

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

Veit, A., Wilber, M. J., and Belongie, S. J. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, pp. 550–558, 2016.

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

Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Behram, F. A., Huang, J., Bai, C., Gschwind, M., Gupta, A., Ott, M., Melnikov, A., Candido, S., Brooks, D., Chauhan, G., Lee, B., Lee, H.-H. S., Akyildiz, B., Balandat, M., Spisak, J., Jain, R., Rabbat, M., and Hazelwood, K. Sustainable ai: Environmental implications, challenges and opportunities, 2022. URL https://arxiv.org/abs/2111.00364.

Xu, M., Yin, W., Cai, D., Yi, R., Xu, D., Wang, Q., Wu, B., Zhao, Y., Yang, C., Wang, S., Zhang, Q., Lu, Z., Zhang, L., Wang, S., Li, Y., Liu, Y., Jin, X., and Liu, X. A survey of resource-efficient llm and multimodal foundation models, 2024. URL https://arxiv.org/abs/2401.08092.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

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

A Ablation: Memory efficiency

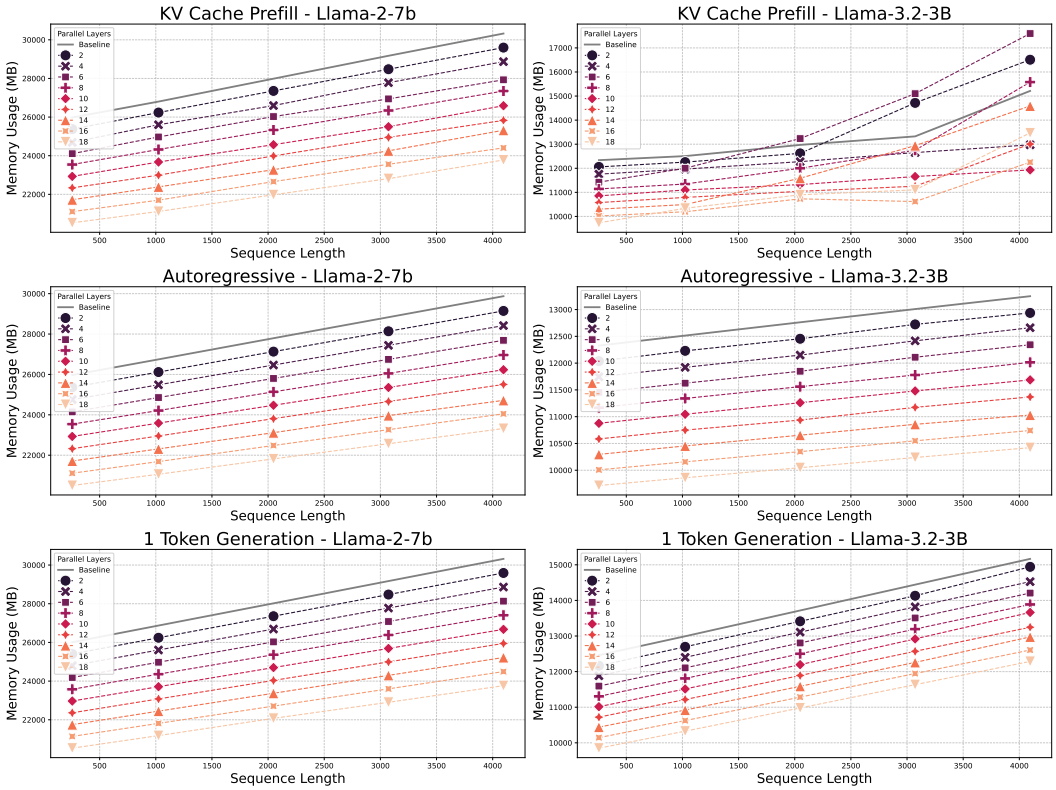


Figure 8: Maxiumum memory usage (Mb) to complete the following inference tasks: KV Cache pre-filling for a given sequence length, autoregressive generation up to the indicated sequence length, and single token generation with a pre-filled KV Cache of the indicated sequence length. The baseline is the 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).

B Ablation: Tokens per second

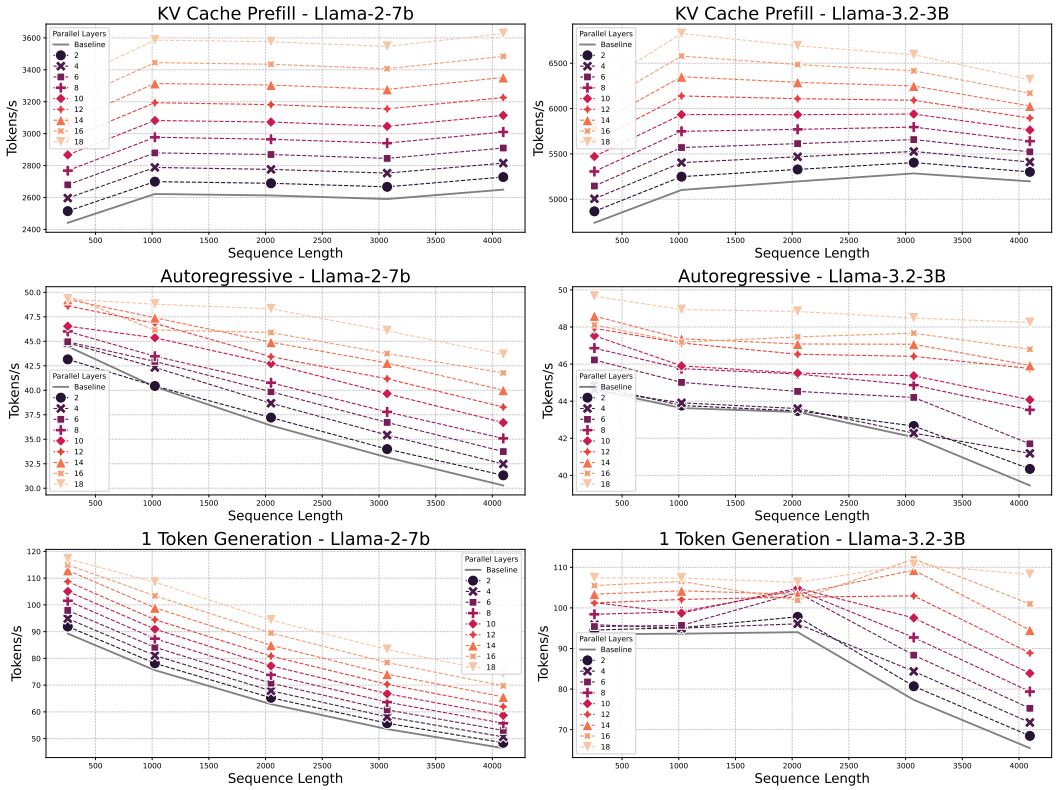


Figure 9: Tokens per second when completing the following inference tasks: KV Cache pre-filling for a given sequence length, autoregressive generation up to the indicated sequence length, and single token generation with a pre-filled KV Cache of the indicated sequence length. The baseline is the original 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.

C Generalization to multiple GPUs

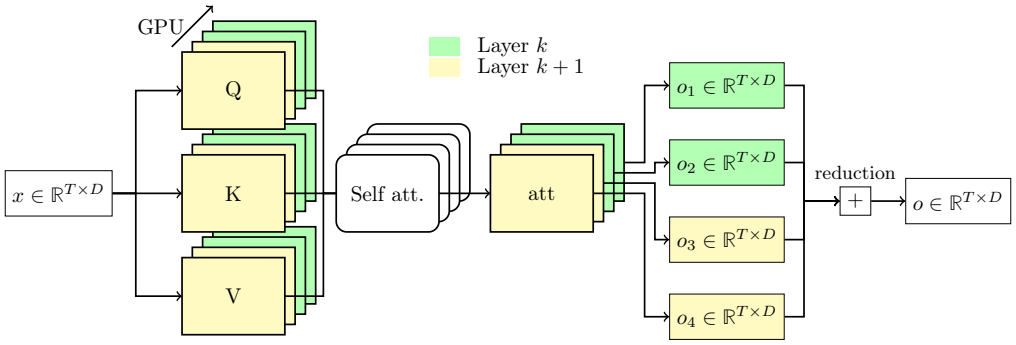


Figure 10: Generalization of Layer Parallelism and Tensor Parallelism. In this case, the figure shows the case for 4 GPUs and 2 layers. The stacked layers represent the tensor parallelism, and the colors indicate the processing of different previously contiguous layers. LP builds on top of TP by assigning the heads from consecutive layers into different GPUs. If there are more GPUs than layers, then each layer will be accelerated using TP, and Q, K, V and $att \in \mathbb{R}^{T \times \frac{2D}{g}}$, where D is the feature dimension and g is the total number of GPUs.