# Crown, Frame, Reverse: Layer-Wise Scaling Variants for LLM Pre-Training

**Andrei Baroian** [1] **Kasper Notebomer** [1] **Max Van Den Boom** [1] **Andres Aranguren** [1]

## Abstract

Transformer-based language models traditionally use uniform (isotropic) layer sizes, yet they ignore the diverse functional roles that different depths can play and their computational capacity needs. Building on Layer-Wise Scaling (LWS) and pruning literature, we introduce three new LWS variants - Framed, Reverse, and Crown - that redistribute FFN widths and attention heads via two or three-point linear interpolation in the pre-training stage.We present the first systematic ablation of LWS and its variants, on a fixed budget of 180M parameters, trained on 5B tokens. All models converge to similar losses and achieve better performance compared to an equal-cost isotropic baseline, without a substantial decrease in training throughput. This work represents an initial step into the design space of layer-wise architectures for pre-training, but future work should scale experiments to orders of magnitude more tokens and parameters to fully assess their potential.

## 1. Introduction

Transformer-based large language models (LLMs) have rapidly become foundational to progress across multiple domains, ranging from natural language understanding and reasoning to vision, robotics, and are regarded as a path towards artificial general intelligence.

Current LLM designs almost universally rely on isotropic architectures - all layers have the same hidden dimensions, number of attention heads, and feed-forward widths. Although this design simplifies implementation and scaling, such uniform parameter allocation may not reflect the most efficient use of model capacity. In practice, different layers of an LLM are known to play different functional roles, with earlier layers capturing local syntactic patterns and later layers responsible for high-level abstraction and reasoning (Jin et al., 2025) (Chen et al., 2024) (Skean et al., 2025).

[1]LIACS, Leiden University, The Netherlands. Correspondence to: Andrei Baroian <m.a.baroian@umail.leidenuniv.nl>.

Therefore, it is reasonable to hypothesize that different layers within the network may benefit from varying levels of computational capacity. A non-uniform, or heterogeneous, distribution of parameters across layers could lead to more efficient utilization of the model's capacity, enhancing performance, all within the constraints of a fixed parameter budget.

With this motivation, Mehta et al. (2024) introduced Layer-Wise Scaling (LWS) in OpenELM, achieving superior performance with half the pre-training tokens compared to similar models, attributing much of this gain to LWS. It assigns fewer parameters to initial layers and linearly increases it toward deeper layers. Despite these promising claims, the specific contribution of Layer-Wise Scaling remains ambiguous due to the absence of targeted ablation studies that isolate its effects from other architectural modifications.

Another component adopted from Mehta et al. (2024) is Grouped Query Attention (GQA), which is tightly coupled with the LWS implementation in that work. Since GQA contributes to efficiency improvements and was used in the original LWS paper, we apply it consistently across all of our experiments.

A parallel line of research has emerged around the improvement of inference efficiency through parameter pruning techniques, which aim to reduce model size and computational cost by removing redundant weights from already trained LLMs (Zhu et al., 2024). Initially, the pruning methods applied isotropic sparsity patterns, treating all layers uniformly (Wang et al., 2024). However, more recent approaches have introduced a diverse range of strategies and algorithms that analyze and quantify the relative importance of parameters throughout the network and pruning layers accordingly. (Jin et al., 2025; Chen et al., 2024; Skean et al., 2025; He et al., 2024)

In this work, we first conduct controlled experiments to isolate and evaluate the effect of (vanilla) Layer-Wise Scaling. Secondly, inspired by the findings from the pruning literature regarding the importance of certain layers, we investigate whether initiating pre-training with architectures resembling the pruned models can enhance performance. Specifically, we explore three novel architectural variations: (i) applying LWS while maintaining maximal size in the first and last layers - Framed LWS; (ii) a reverse framed

LWS strategy, beginning with larger initial layers that gradually reduce in size; and (iii) a crown-shaped configuration, reflecting empirical evidence suggesting central layers are of greatest importance, used together with framing.

Our experiments are conducted on a 180M-parameter model, built upon the OLMo-core repository, and trained on a corpus of 5 billion tokens. Although the experiments are based on a 190M parameter model, this scale is increasingly recognized as a reference setting to explore architectural efficiency under realistic computing constraints (Mehta et al., 2024). Small LLM models serve as reliable testbeds for evaluating generalization of architectural innovations (Hoffmann et al., 2022a). Prior scaling studies have shown that techniques yielding efficiency gains in small models often extrapolate to larger scales when governed by principled design rules. Consequently, understanding the isolated effect of layer-wise scaling at this size provides evidence for its potential application in larger models, where sample efficiency translates directly into reduced training costs and environmental impact (Kaplan et al., 2020a).

The code for our experiments is available open-source at `https://github.com/baroian/OLMo-custom`.

In Section 2 we begin by reviewing related work on heterogeneous architectures and motivating our four LWS profiles. In Section 3, we formalize the Layer-Wise Scaling framework and detail its four variants. Section 3 also outlines the model architectures, training setup, and datasets. Section 4 presents the results, while Section 5 concludes with a discussion of limitations and future research.

## 2. Related Work

This section will explain the origin of Layer-Wise Scaling, present relevant pruning literature that inspired the new variants. It is followed by an explanation of Group Query Attention (GQA) and how it is applied in LWS.

### 2.1. Layer Wise Scaling (LWS)

Layer-Wise Scaling (LWS) is an architectural strategy for Transformer models that deviates from traditional isotropic scaling - where all layers share identical configurations (Kaplan et al., 2020a; Hoffmann et al., 2022a)- by systematically varying structural parameters such as the number of attention heads $n_{\text{h}}^{(i)}$ and feed-forward hidden dimensions $d_{\text{FFN}}^{(i)}$ on a per-layer basis (Mehta et al., 2024; 2021). The core premise is that different layers fulfill distinct functional roles and thus require different computational capacities. OpenELM (Mehta et al., 2024) implements LWS by linearly interpolating scaling factors $n_{\text{h}}^{(i)}$ and $d_{\text{FFN}}^{(i)}$, becoming wider as it goes deeper. In other words, it linearly increases the number of parameters of each layer as it goes deeper. This strategic allocation of a model's parameter budget is motivated by the pursuit of optimized parameter allocation, enhanced model accuracy, and improved data efficiency as demonstrated by OpenELM, which achieves superior performance with twice fewer pre-training tokens compared to uniformly scaled models like OLMo (Mehta et al., 2024).

Although the claims are promising, the exact role of Layer-Wise Scaling remains unclear, as the authors have not presented targeted ablation studies to separate its impact from other architectural changes, revealing a gap in the literature. Despite thorough search efforts, we have found no other studies that applied the concept of layer-wise scaling during pre-training.

### 2.2. Pruning Literature

Pan et al. (2025) introduced Layer-Wise Adaptive Pruning which computes the relative importance of each decoder layer to decide which parameters to prune, and showed it is "extremely effective" in experiments on 32-layer LLaMA-3.1-8B model. Their pruned model does not prune the first and last layer; second layer starts at half the maximum parameters and linearly **decreases** to less than a quarter of the initial size. This suggests that allocating more computation to initial layers is better. The same conclusion is derived by Huang et al. (2025) who took a theoretical perspective and proved that cutting out early layers leads to a "reconstruction error explosion" where the errors from pruning initial layers compound over the rest of the network. This would imply that this pruned architecture is relevant only for pruning. Nevertheless, we consider it worthwhile to investigate a variant of LWS that allocates more parameters to the early layers, with a linearly decreasing allocation toward the later parts of the network.

Askari et al. (2025) introduced Influence Functions to estimate the layer quality and experiment on 32 layers Mistral 7B and Gemma 7B. They use this method in both pruning and expert allocation (in Mixture of Experts model), and find that the middle layers show the greatest contribution to performance. Gao et al. (2025) investigate the importance of layers in the context of Mixture of Experts LORA fine-tuning, and found superior performance by allocating more experts to the middle layers.

The most relevant paper comes from He et al. (2024). They **pre-train** a 7B parameter 28 layers model on 500B tokens while keeping track of the similarity score throughout the training (see Fig. 1). They operate on the premise that redundant transformer blocks produce outputs highly similar to their inputs; they prune (post-training) by dropping attention layers, MLP layers, or entire transformer blocks. They concluded that deeper layers exhibit low importance and drop the number of blocks almost linearly decreasing from the initial to the final layers, but keeping the last layer almost

intact. Importantly, their pruning strategy (we refer to it as Reverse LWS) does not completely reflect the findings of the similarity score (see Fig. 1). In their visualization of the similarity score throughout the training, the importance of layers peaks in the middle of the network between layers 8 and 20 while the first and final layers are still of the highest importance. This inspired the Crown LWS variant.

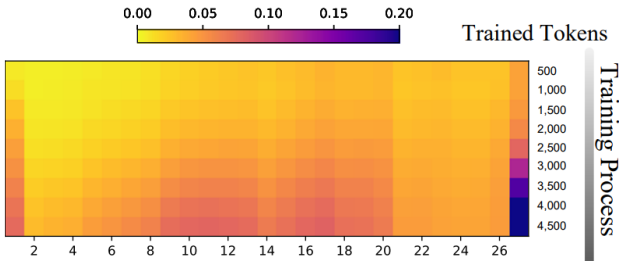


*Figure 1.* Figure from He et al. (2024) showing similarity score across training. Block number on x-axis, darker shades indicate higher block importance

Another insight from the same paper comes from the shifting of similarity score during pre-training, as it starts relatively flat across all layers and gradually transitions into a "crown"-shaped pattern toward the end of training.

It is important to note that in post-training compression methods, the importance of the layer is typically computed with respect to a specific dataset. Tan et al. (2024) empirically showed that the contribution of individual layers varies between datasets.

Inspired by these research papers, we develop and investigate Framed LWS, Reverse LWS, and Crown LWS which will be explained in Section 3.

### 2.3. Group Query Attention (GQA)

Group Query Attention (GQA) aims to mitigate the substantial memory bandwidth and capacity demands of the Key-Value (KV) cache in standard Multi-Head Attention (MHA) during autoregressive inference (Ainslie et al., 2023). GQA achieves this by dividing the total $N_q$ query heads into $G$ distinct groups, where the query heads within each group share a single Key ($K$) and Value ($V$) projection. This approach reduces the KV cache size from $N_q \times L \times d_k \times 2 \times$ precision in MHA to $G \times L \times d_k \times 2 \times$ precision, where $L$ is the sequence length and $d_k$ is the key / value head dimension. Thus, GQA offers a tunable trade-off between the MHA model quality (equivalent to GQA when $G = N_q$) and the aggressive reduction of KV cache of Multi-Query Attention (MQA) (Shazeer, 2019) (equivalent to GQA when $G = 1$), leading to improved inference efficiency while largely preserving performance. As illustrated in Figure 2, GQA groups query heads such that each group shares a single key and value projection, reducing memory overhead. This visual demonstrates how GQA interpolates between the full flexibility of MHA and the efficiency of MQA.

**Combining Layer-Wise Scaling with GQA**. The number of key/value heads $G$ is kept *fixed* in every layer. Scaling the attention therefore occurs solely by increasing the number of query heads $H_q$, while the key and value heads remain $H_k = H_v = G$. Because each KV-group serves exactly $H_q/G$ query heads, the query-head count must satisfy $H_q \bmod G = 0$. In all the experiments $G = 4$. This inhibits the impact of LWS on scaling attention as there will be fewer and rougher transitions (eg: from 8 heads to 12 heads). However, we decided to use it in all experiments, as it was utilized in the original LWS study.

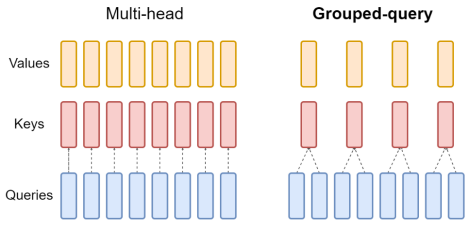


*Figure 2.* Grouped-Query Attention (GQA): Each group of query heads shares key and value projections, bridging MHA and MQA for improved efficiency.

## 3. Methods

In this section, the architecture of the baseline model is explained, followed by details on each of the four variants. Then the framework for building the feedforward and attention layers is presented, concluding with a description of the evaluation procedure.

### 3.1. Base Model & Training Procedure

To evaluate the effect of Layer-Wise Scaling and its variants in isolation, we implement these changes on the OLMo 2 190M as it is part of a clean and well-maintained codebase. The complete model specifications are provided in (OLMo et al., 2025). Worth mentioning, the model is a dense decoder-only transformer (Vaswani et al., 2023) with no biases, SwiGLU activation function, Rotary positional embeddings (RoPE), RMSNorm and QK-norm. Table 1 shows the specific hyperparameters used for the baseline. Macro-batch size was chosen to fit a single GPU. Batch size (must be a multiple of macro-batch size and GPU count) and learning rate were chosen based on (Biderman et al., 2023; Brown et al., 2020; Kaplan et al., 2020b) who utilize a batch size of 0.5M tokens and learning rate of 6e-4. Four KV heads create 3 GQA groups.

| Hyper-parameter | Value |
|---|---|
| Layers ($L$) | 12 |
| Hidden size ($d_{\text{model}}$) | 768 |
| Attention heads ($H$) | 12 |
| KV heads | 4 |
| Max sequence length | 1 024 |
| Macro batch size | 48 sequences |
| Global batch size | 384 sequences |
| Training steps | 1 3000 |
| Learning rate | $6 \times 10^{-4}$ |
| Optimizer | AdamW |
| Tokenizer | gpt neox olmo dolma |
| Vocabulary size | 50 279 |

*Table 1.* Training hyper-parameters for Baseline 12 Layers

We train all the models on 5 billion tokens of high-quality web pages. The tokens were sampled from a version of DataComp for Language Models (DCLM) (Li et al., 2025) filtered for quality by OLMo Team as part of DOLMino dataset mix (Dolmino Mix 1124). A held-out set of 10 million tokens was created for evaluation.

### 3.2. Layer Wise Scaling and Variants

Using the existing literature, we test four Layer-Wise Scaling variants (see Fig. 3):

- Vanilla LWS - as used in (Mehta et al., 2024), linearly increasing the size of layers
- Framed LWS - same as Vanilla LWS but keeping the first layer at maximum dimension
- Reverse LWS - starting with many parameters in early layers and linearly decreasing until the end; it is framed i.e. last layer has the maximum dimension.
- Crown LWS - achieving maximum dimensions in the middle layers; it is framed, i.e. the first and last layers have the maximum dimension.

**Vanilla Layer-Wise Scaling** is implemented based on the methodology proposed by (Mehta et al., 2024). The two components of the transformer architecture that are scaled are the FNN and the attention mechanism. The standard transformer architecture, as used in OLMo, with a given depth N, scales the FFN layer using the following formula $d_{fnn} = \beta * d_{model}$. Here, $d_{fnn}$ is the dimensionality of the FNN layer for the earch transformer block, $\beta$ is a fixed scalar and $d_{model}$ is the input dimension of the transformer blocks. For layerwise scaling, this is changed to:

$$d^i_{ffn} = \beta^i * d_{model} \tag{1}$$

$$\beta^i = \beta_{start} + \frac{(\beta_{end} - \beta_{start}) * i}{N - 1} \tag{2}$$

Here $d^i_{fnn}$ is the dimensionality of the FFN for the $i$-th transformer of the $N$ total, $0 \leq i \leq N$. Every layer is scaled by $\beta^i$, which is calculated using a linear interpolation between a given $\beta_{start}$ and $\beta_{end}$.

For multi-headed attention (MHA) the default transformer has a given number of attention heads, $n_h$. The dimensionality of each head $d_h$ can then be expressed as $d_h = \frac{d_{model}}{n_h}$.

In layer-wise scaling, the dimensionality of each attention heads is kept equal for all transformer layers; instead, the number of attention heads is scaled. The exact formulation is as follows:

$$n^i_h = \alpha^i \cdot \frac{d_{model}}{d_h} \tag{3}$$

$$\alpha^i = \alpha_{start} + \frac{(\alpha_{end} - \alpha_{start}) * i}{N - 1} \tag{4}$$

Similarly to FNN, the scalar for each layer, $\alpha^i$, is a linear interpolation between a given $\alpha_{start}$ and $\alpha_{end}$. The concatenation of all the head of attention is now different in size than the dimension of the model, so the linear projection is used to scale the output of the MHA back to $d_{model}$.

To achieve **Reverse LWS**, the same formulation and interpolation as LWS is used, but with the constraint that $\alpha_{start} \geq \alpha_{end}$ and $\beta_{start} \geq \beta_{end}$. This results in a parameter allocation scheme where the number of parameters per transformer decreases as the model becomes deeper.

Another variation is **Framed LWS**, where the sizes of the first and last transformers are fixed, and therefore unaffected by LWS. The propsed method achieves this by setting the

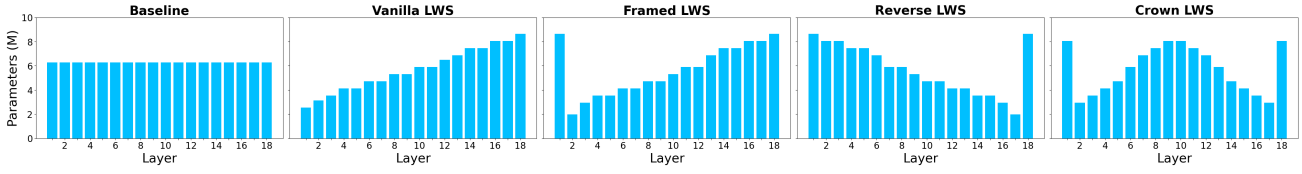


*Figure 3.* **Layer-Wise Scaling (LWS) variants used in our study.** Parameter-allocation profiles for 18 layers : *Baseline* is an uniform (isotropic) model; (*a*) *Vanilla LWS* linearly increases from shallow to deep; (*b*) *Framed LWS* keeps the first and last layers at maximum size while linearly scaling; (*c*) *Reverse LWS* more parameters allocated to initial layers, decreasing it toward later layers, using framing; (*d*) *Crown LWS* parameter count peaks in the middle layers, using framing. All models have the same total parameter count of 180M.

scalars for the first and last transformer blocks using $\alpha = max(\alpha_{start}, \alpha_{end})$ and $\beta = max(\beta_{start}, \beta_{end})$.

The last variant is **Crown LWS**. For these models, more parameters are assigned to the middle transformer blocks compared to the first and final blocks. The same linear interpolation for $\alpha$ (4) and $\beta$ (2) is used. To achieve the crown scaling, the interpolation is now done between three scalars instead: $[\alpha_{start}, \alpha_{middle}, \alpha_{end}]$ for MHA and $[\beta_{start}, \beta_{middle}, \beta_{end}]$ for FFN. $\alpha_{start}$ and $\alpha_{end}$ represent the scalars for the first and last transformer blocks, respectively. The additional $\alpha_{middle}$ scalar controls the size of the middle transformer block. The interpolation is performed first between $\alpha_{start}$ and $\alpha_{middle}$, and then between $\alpha_{middle}$ and $\alpha_{end}$ to calculate the scalars for all transformer blocks. The same interpolations are made for $\beta$.

### 3.3. Evaluation

The evaluation of the LWS variants will be made based on validation perplexity. This part will motivate our choice and explain why reporting results on benchmarks or generated text is infeasible.

Biderman et al. (2023) trained a range of model sizes from 70M to 12B parameters and share their results on common benchmarks suited for the small-scale pre-training phase. For SciQ benchmark (Johannes Welbl, 2017) All the models show no growth signal until the 10 billion tokens from where the accuracy increases sharply from 0.2 to 0.6-0.8 until the 100B tokens mark, followed by a plateau or slow increase until the full 300B token training. Other benchmarks show similar growth while some display no discernible learning signal. This together with our early experiments on 4B tokens which did not showed any learning signal on any benchmarks, led us to abandon evaluating the LWS methods on benchmarks and instead focus on perplexity.

Attempts to introduce changes in architectures for language models use validation perplexity (Zhou et al., 2024) as well as training loss (Du et al., 2024). In their analysis of hyperparameter configurations, OLMo et al. (2025) adopted training loss as the reference metric. Moreover, the conclusions of (Kaplan et al., 2020a) and (Hoffmann et al., 2022b) are based on training cross-entropy loss. These give us confidence that validation perplexity is a good measurement for evaluating the LWS's performance.

Perplexity measures a model's uncertainty by reporting the geometric mean of the inverse probabilities it assigns to each true next token (Jurafsky & Martin, 2025) , defined as $\mathrm{PPL}(W) = P(w_{1:N})^{-1/N}$ and computed in our context as $\mathrm{PPL} = \exp\big(\mathcal{L}_{\mathrm{CE}}\big)$.

## 4. Experiments & Results

The initial OLMo2-190M was chosen as the baseline, with all layers having the same dimension, but it has only 12 layers. With so few layers, the interpolation occurring in LWS would produce noticeable jumps rather than a smooth progression, especially if we are using framing where two fewer layers are scaled. As a response to this, we increased the number of layers to 18 for all models including the baseline, keeping the number of parameters the same. In other words, we prefer to exchange depth for width in our experiments, although it might affect performance as "in general, deeper models have lower perplexity". (Petty et al., 2024). We still run the baseline and vanilla LWS with 12 layers and compare them with the 18 layers variants.

To account for the randomness in the training process that affects the results, we had the option to either run each experiment two times or run the baseline five times giving our compute constraints. We chose the latter and ran the 18-layer baseline five times, recording the standard deviation to determine statistical significance of the results.

Table 2 lists all the models used in our experiments including the parameter count. All variants include GQA, that is why the*baseline 12L* has 180M tokens instead of the 190M from olmo2-190M configuration.

All experiments were run on Snellius (SURF, 2025), on a single node with 4 H100 GPUs, each running for approximately three hours.

### Results

Table 3 shows the tokens per second, validation loss and validation perplexity for all models. First, we will look at the baseline against vanilla LWS on both 12 layers and 18 layers.

*Table 2.* Architecture details, scaling factors, framing setting, and parameter counts.

| MODEL | $n_{\text{LAYERS}}$ | FNN_SCALAR | QKV_SCALAR | FRAMING | PARAMS (M) | NON-EMBED (M) |
|---|---|---|---|---|---|---|
| BASELINE 12L | 12 | [4.0, 4.0] | [1.0, 1.0] | FALSE | 181.1 | 142.5 |
| VANILLA LWS 12L | 12 | [2.0, 5.3] | [0.5, 2.0] | FALSE | 178.8 | 140.1 |
| BASELINE 18L | 18 | [2.5, 2.5] | [0.75, 0.75] | FALSE | 183.5 | 144.8 |
| VANILLA LWS 18L | 18 | [1.0, 4.0] | [0.5, 1.0] | FALSE | 179.7 | 141.1 |
| FRAMED LWS | 18 | [0.5, 4.0] | [0.5, 1.0] | TRUE | 179.4 | 140.7 |
| REVERSE LWS | 18 | [4.0, 0.5] | [1.0, 0.5] | TRUE | 179.4 | 140.7 |
| CROWN LWS | 18 | [0.5, 3.8, 0.5] | [0.5, 1.0, 0.5] | TRUE | 181.9 | 143.3 |

It can be clearly observed in Figure 4 that the difference is insignificant at the 12 layers, but it is meaningful at the 18 layer models, vanilla LWS having a 6% better perplexity. The results on 12 layers are expected as the transitions from one layer to another are rough, showing that LWS works only on deeper networks. We can therefore hypothesize that on bigger models with more layers, the transition of LWS would be smoother, thus its benefits greater.

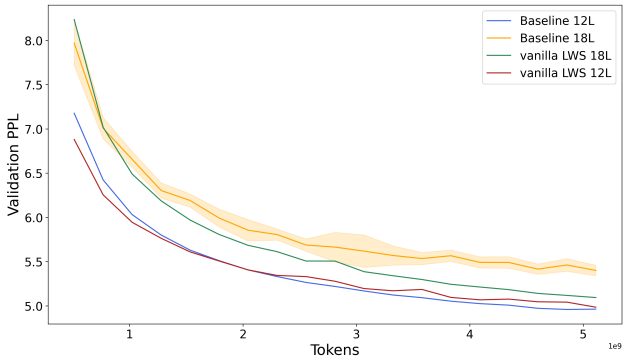


*Figure 4.* Validation Perplexity for LWS vs Baseline at 12 and 18 layers

Figure 7 shows the results of all the variants. Unexpectedly, all the variants performed almost the same, all significant better than the baseline. To exclude the possibility that this convergence arose from implementation error, for example, inadvertently training of identical architectures, we manually inspected the run logs and confirmed that the correct dimensions were instantiated in every run. Framed LWS though records a significant higher final perplexity than the other variants, but we cannot conclude that it is worst; looking at the figure, it had a bump in PPL right before the final steps, bump that can also be observed to vanilla LWS at around 2.7B tokens and Crown LWS at 4B tokens, and both of them recovered sharply. Results suggest that the presence of heterogeneity, not its exact shape, matters.

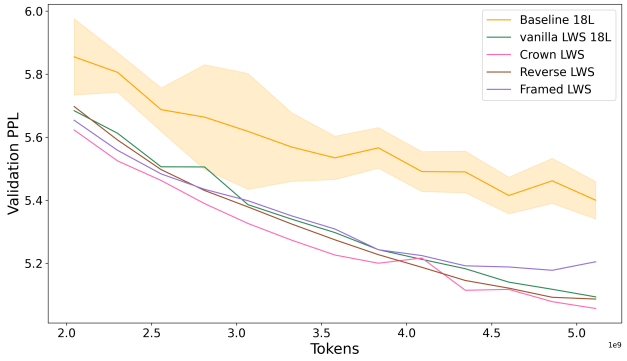


*Figure 5.* Validation Perplexity on LWS variants, all 18 Layers; Zoomed in for readability by displaying results from 2B token on

Looking at the tokens per second in table 3, the LWS variants do not slow down the training process for the 18L models, but it does show a notable decrease in TPS at the 12 layers. The $\approx 10\%$ drop in training speed training speed showed in the 12-layer models is explained as the uniform model repeats exactly the same matrix size in every feed-forward block, allowing the GPU to reuse the same low-level kernel throughout each training step. The LWS 12L model, by contrast, introduces different layer sizes and every time the size changes the GPU has to load a different kernel, adding a setup overhead. We hypothesize that the slow-down does not appear when comparing Baseline 18L with the LWS variants as we have created the configuration for the baseline and chose scalars without taking into consideration effective use of GPUs.

Training Cross Entropy Loss can be seen in Appendix A

*Table 3.* Final validation perplexity (PPL) for all models after step $\geq 1000$. Standard deviation of Baseline 18L on last step is 0.0595 TPS = Tokens per Second processed (i.e. how fast it trains)

| MODEL | TPS | VAL. LOSS | VAL. PPL |
|---|---|---|---|
| BASELINE 12L | **146K** | 1.6018 | **4.962** |
| VANILLA LWS 12L | 138K | 1.6062 | 4.984 |
| BASELINE 18L | 124K | 1.6864 | 5.400 |
| VANILLA LWS 18L | 124K | 1.6279 | 5.093 |
| FRAMED LWS | 123K | 1.6490 | 5.205 |
| REVERSE LWS | 123K | 1.6266 | 5.087 |
| CROWN LWS | 121K | 1.6206 | **5.057** |

## 5. Discussion

This work represents an initial step into the design space of layer-wise heterogeneous architectures for pre-training.

All LWS variants have showed improvement up to 5% in performance compared to the uniform baseline. This reveals that any heterogeneous allocation of parameters is preferable to a uniform one at equal compute, yet the exact profile (vanilla, framed, reverse, or crown) matters little. In other words, LWS reallocates rather than creates representational capacity, yielding incremental gains rather than transformative improvements.

This incremental benefit is far from the expectations created by Mehta et al. (2024). Based on our experiments we conclude that LWS alone does not result in better or similar performance than a baseline trained on double the tokens, as claimed in their paper. The performance gains claimed in their study therefore arises from synergies between LWS and other components in the OpenELM training recipe.

Many post-training compression techniques (He et al., 2024; Huang et al., 2025; Pan et al., 2025; Gao et al., 2025; Askari et al., 2025) have shown that transformer-based models can prune a significant number of parameters with only minor

losses in downstream performance, creating efficiency gains at inference. We have questioned if we could bring those gains at the pre-training stage. A direct analogy would involve comparing a large uniform model to a slightly smaller LWS-based model and assessing whether the performance remains comparable. Instead, we chose to fix model size across both configurations, hypothesizing that Layer-Wise Scaling (LWS) could lead to improved performance at equivalent parameter count. A limitation of this setup is that it does not allow us to directly assess whether the observed efficiency improvements align quantitatively with those reported in post-training pruning studies. We nevertheless observe consistent performance gains across all LWS configurations, suggesting that even during pre-training, careful redistribution of parameters can enhance model effectiveness, resonating with the insights from pruning literature.

### 5.1. Weakneseess

The limited scale of our experiments, particularly the 5B-token training corpus, introduces uncertainty to the conclusions. Training on 100 billion tokens would have enabled evaluation on downstream benchmarks and provided a more rigorous assessment of the proposed approaches. ;it would also enable evaluation of the generated text, which is incoherent at our scale.

Another break point of our experiments could be the unusual low training loss as well as validation perplexity of 5 and CE loss while showing a healthy training trend - loss going gradually down with sporadic bumps. All the papers cited that pre-train LLMs have a final validation perplexity between 10 and 15, models that are 10 to 50 times bigger and trained on 60 to 500 times more tokens. The content, size and token distribution of the data set were examined, and no irregularities or sources of concern were observed. A possible reason is the data distribution as the training set comes from the same data source, a more diverse data mix would result in a higher perplexity and a higher generalization. A future investigation could test this hypothesize by using a different validation set outside of current distribution.

Another weakness comes from the compute-constraints, as running all experiments five times would give a clearer delimitation between the models performance. The LWS variants could have higher standard deviation compared to the baseline.

### 5.2. Future Work

Our results highlight LWS's potential, but further exploration is needed to fully understand its benefits across model and dataset sizes. At first, future work should test LWS variants on multiple, bigger models, from 190M to 7B parameters model, the size where it becomes clear if a change transfers to larger models, and to train on a much bigger corpus starting with 100B tokens and reach OLMo2-7B scale of 3B tokens.

## 6. Conclusion

Our experiments revisited Layer-Wise Scaling (LWS) for transformer language models, introducing three pruning-inspired variants (Framed, Reverse, and Crown LWS) under a fixed 180 M-parameter budget, and showed that all LWS variants improve in validation perplexity over an isotropic baseline. We found that LWS alone does not replicate the two times data efficiency gains previously claimed by OpenELM. While these findings underscore the value of heterogeneity, our study is limited by a modest 5 B-token corpus, evaluation solely via perplexity, and single-run training, suggesting that future work should involve larger-scale pre-training ($\geq$ 100 B tokens, $\geq$ 7 B parameters).

## Acknowledgements

We would like to express our sincere gratitude to Dr. Hazel Doughty for offering this seminar and for her support throughout the project. We also thank the Teaching Assistants, Andrius Bernatavicius and Luc Sträter, for helping to create an engaging learning environment.

We are grateful to Prof. Dr. Rob V. van Nieuwpoort for granting us access to Snellius compute resources and placing his trust in our work.

Finally, we would like to thank Dirk Groeneveld, Principal Researcher at the Allen Institute for AI, whose early advice helped us choose a focused and feasible topic, and avoid overly ambitious or saturated research directions.

## References


Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Askari, H., Gupta, S., Wang, F., Chhabra, A., and Chen, M. Layerif: Estimating layer quality for large language models using influence functions, 2025. URL https://arxiv.org/abs/2505.23811.

Biderman, S., Schoelkopf, H., Anthony, Q., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., Skowron, A., Sutawika, L., and van der Wal, O. Pythia: A suite for analyzing large language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Chen, N., Wu, N., Liang, S., Gong, M., Shou, L., Zhang, D., and Li, J. Is bigger and deeper always better? probing llama across scales and layers, 2024. URL https://arxiv.org/abs/2312.04333.

Dolmino Mix 1124, year = 2024, n. . O. h. . https://huggingface.co/datasets/allenai/dolmino-mix-1124. Dolmino mix 1124.

Du, W., Luo, T., Qiu, Z., Huang, Z., Shen, Y., Cheng, R., Guo, Y., and Fu, J. Stacking your transformers: A closer look at model growth for efficient llm pre-training, 2024. URL https://arxiv.org/abs/2405.15319.

Gao, C., Chen, K., Rao, J., Liu, R., Sun, B., Zhang, Y., Peng, D., Guo, X., and Subrahmanian, V. MoLA: MoE LoRA with layer-wise expert allocation. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5097–5112, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.284. URL https://aclanthology.org/2025.findings-naacl.284/.

He, S., Sun, G., Shen, Z., and Li, A. What matters in transformers? not all attention is needed, 2024. URL https://arxiv.org/abs/2406.15786.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022a.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022b. URL https://arxiv.org/abs/2203.15556.

Huang, W., Zhang, Y., Zheng, X., Chao, F., and Ji, R. Determining layer-wise sparsity for large language models through a theoretical perspective, 2025. URL https://arxiv.org/abs/2502.14770.

Jin, M., Yu, Q., Huang, J., Zeng, Q., Wang, Z., Hua, W., Zhao, H., Mei, K., Meng, Y., Ding, K., Yang, F., Du, M., and Zhang, Y. Exploring concept depth: How large language models acquire knowledge and concept at different layers?, 2025. URL https://arxiv.org/abs/2404.07066.

Johannes Welbl, Nelson F. Liu, M. G. Crowdsourcing multiple choice science questions. 2017.

Jurafsky, D. and Martin, J. H. Chapter 3: N-gram language models. In *Speech and Language Processing*. 3rd ed. (draft) edition, 2025. Draft dated 12 Jan 2025. URL: https://web.stanford.edu/~jurafsky/slp3/3.pdf.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020a.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020b. URL https://arxiv.org/abs/2001.08361.

Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., Garg, S., Xin, R., Muennighoff, N., Heckel, R., Mercat, J., Chen, M., Gururangan, S., Wortsman, M., Albalak, A., Bitton, Y., Nezhurina, M., Abbas, A., Hsieh, C.-Y., Ghosh, D., Gardner, J., Kilian, M., Zhang, H., Shao, R., Pratt, S.,

Sanyal, S., Ilharco, G., Daras, G., Marathe, K., Gokaslan, A., Zhang, J., Chandu, K., Nguyen, T., Vasiljevic, I., Kakade, S., Song, S., Sanghavi, S., Faghri, F., Oh, S., Zettlemoyer, L., Lo, K., El-Nouby, A., Pouransari, H., Toshev, A., Wang, S., Groeneveld, D., Soldaini, L., Koh, P. W., Jitsev, J., Kollar, T., Dimakis, A. G., Carmon, Y., Dave, A., Schmidt, L., and Shankar, V. Datacomp-lm: In search of the next generation of training sets for language models, 2025. URL https://arxiv.org/abs/2406.11794.

Mehta, S., Ghazvininejad, M., Iyer, S., Zettlemoyer, L., and Hajishirzi, H. Delight: Deep and light-weight transformer, 2021. URL https://arxiv.org/abs/2008.00623.

Mehta, S., Sekhavat, M. H., Cao, Q., Horton, M., Jin, Y., Sun, C., Mirzadeh, I., Najibi, M., Belenko, D., Zatloukal, P., et al. Openelm: An efficient language model family with open training and inference framework. *arXiv preprint arXiv:2404.14619*, 2024.

OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., Lambert, N., Schwenk, D., Tafjord, O., Anderson, T., Atkinson, D., Brahman, F., Clark, C., Dasigi, P., Dziri, N., Guerquin, M., Ivison, H., Koh, P. W., Liu, J., Malik, S., Merrill, W., Miranda, L. J. V., Morrison, J., Murray, T., Nam, C., Pyatkin, V., Rangapur, A., Schmitz, M., Skjonsberg, S., Wadden, D., Wilhelm, C., Wilson, M., Zettlemoyer, L., Farhadi, A., Smith, N. A., and Hajishirzi, H. 2 olmo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

Pan, R., Wang, B., Diao, S., Pan, X., Zhang, J., Pi, R., and Zhang, T. Adapt-pruner: Adaptive structural pruning for efficient small language model training, 2025. URL https://arxiv.org/abs/2502.03460.

Petty, J., van Steenkiste, S., Dasgupta, I., Sha, F., Garrette, D., and Linzen, T. The impact of depth on compositional generalization in transformer language models, 2024. URL https://arxiv.org/abs/2310.19956.

Shazeer, N. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Skean, O., Arefin, M. R., Zhao, D., Patel, N., Naghiyev, J., LeCun, Y., and Shwartz-Ziv, R. Layer by layer: Uncovering hidden representations in language models, 2025. URL https://arxiv.org/abs/2502.02013.

SURF. Snellius supercomputer. https://www.surf.nl/en/compute/snellius-supercomputer, 2025. Accessed: 2025-06-30.

Tan, Z., Dong, D., Zhao, X., Peng, J., Cheng, Y., and Chen, T. Dlo: Dynamic layer operation for efficient vertical scaling of llms, 2024. URL https://arxiv.org/abs/2407.11030.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Wang, W., Chen, W., Luo, Y., Long, Y., Lin, Z., Zhang, L., Lin, B., Cai, D., and He, X. Model compression and efficient inference for large language models: A survey, 2024. URL https://arxiv.org/abs/2402.09748.

Zhou, Y., Du, N., Huang, Y., Peng, D., Lan, C., Huang, D., Shakeri, S., So, D., Dai, A., Lu, Y., Chen, Z., Le, Q., Cui, C., Laudon, J., and Dean, J. Brainformers: Trading simplicity for efficiency, 2024. URL https://arxiv.org/abs/2306.00008.

Zhu, X., Li, J., Liu, Y., Ma, C., and Wang, W. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577, 11 2024. ISSN 2307-387X. doi: 10.1162/tacl_a_00704. URL https://doi.org/10.1162/tacl_a_00704.

## A. Training Loss

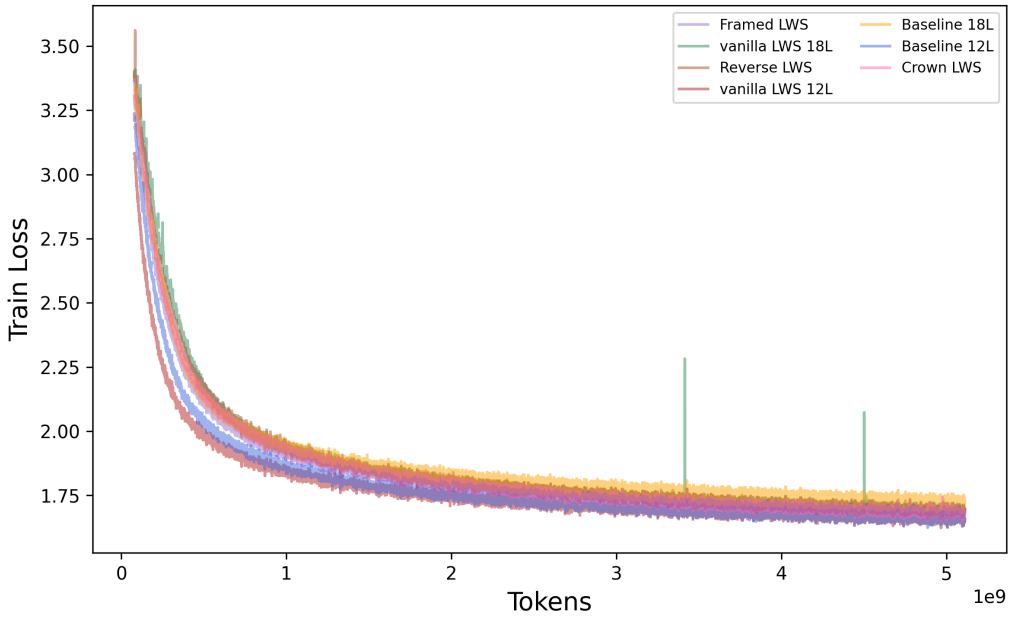


*Figure 6.* Training Cross Entropy Loss

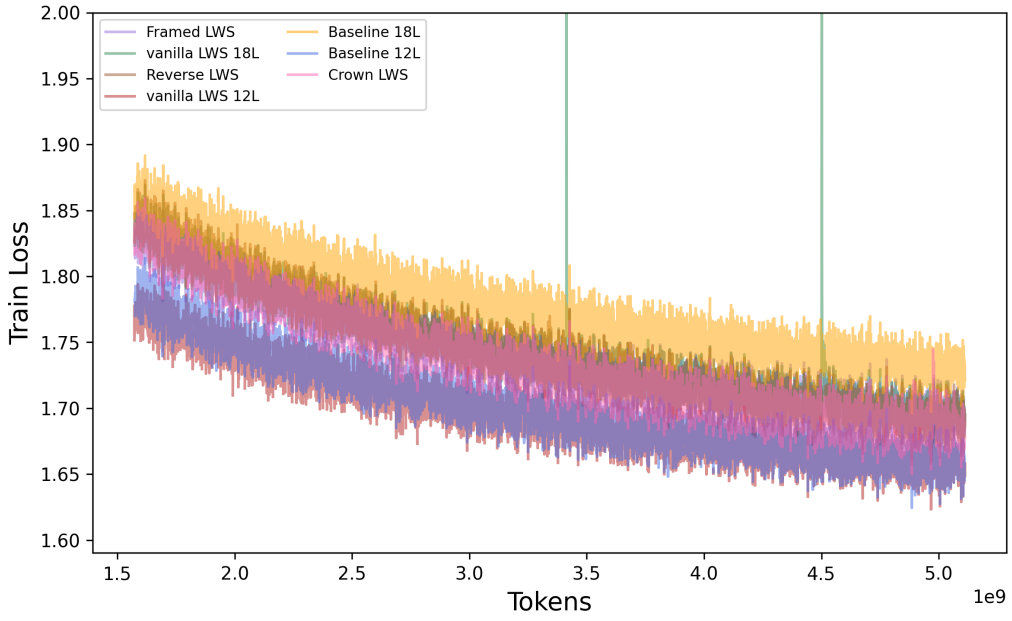


*Figure 7.* Training Cross Entropy Loss Zoomed in