

Impact of Sequence Length on Transformer Training

Mikuláš Vanoušek
EPFL

Oguz Kaan Yüksel
Theory of Machine Learning Lab
EPFL

Abstract—We investigate how context length influences scaling laws and compute efficiency in transformer language models. Training 1–11M-parameter models on matched token budgets across context lengths 128–512, and evaluating all models on 512-token inputs, we find that context length systematically alters fitted scaling-law exponents. In the undertrained regime, shorter-context training is more compute-efficient: 128-token runs outperform 512-token runs at equal FLOPs. Longer-context models eventually overtake once sufficiently trained, exploiting long-range dependencies. Our results show that context length is a meaningful missing variable from current scaling-law formulations and should be explicitly considered when designing training strategies.

I. INTRODUCTION

Scaling-law studies[1], [2] have shown that the validation loss of **large language models (LLMs)** can be accurately modeled as a function of two quantities: the number of model parameters N and the number of training tokens D . These works fit a power-law form in which validation loss $\mathcal{L}$ decreases predictably as N and D increase.

$$\mathcal{L}(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta} \quad (1)$$

However, the exact exponents α and β differ widely between the two landmark studies (see Table IV). This disagreement suggests that important drivers of training dynamics are missing from current formulations.

One key omission is that these classical scaling laws treat the training token count D as the sole data-related factor, without considering the context length used during training. Modern LLMs are trained with increasingly long sequence lengths, and these long contexts affect learning in at least two meaningful ways. First, longer sequences expose the model to more cross-token dependencies. Second, training on long contexts is more computationally expensive due to the quadratic complexity of self-attention, even when the total number of processed tokens is held constant.

Consider two training regimes that run for the **same number of batches**, where each batch contains the **same total number of tokens**, but the composition of those batches differs. For instance, a batch might contain 100 sequences of length 512 (regime A) or 800 sequences of length 64 (regime B). Although both batches have the same token count, regime A will be more computationally demanding due to the quadratic complexity of attention. — and it is unclear whether this extra cost yields proportionally better generalization.

In this work, we revisit scaling laws through the lens of context length. We reproduce classical parameter- and data-scaling results while systematically varying the context length used during training. By holding the total number of tokens constant across runs and modifying only the sequence length, we can isolate the contribution of context length to both compute requirements and final model performance. Understanding this effect is increasingly important, as training frontier-scale models now costs hundreds of millions to billions of dollars, and inefficiencies in sequence-length choice translate directly into wasted compute.

The authors of [3] observe that training on shorter sequences early in the process can improve the final performance of the model. However, they only focus on how to order the data and treat short documents as an inconvenience. We question this assumption and investigate whether training on shorter documents, especially early in the training, can be more effective.

Our results provide empirical evidence that context length meaningfully shifts the scaling behavior of LLMs, especially in the undertrained regime. These findings highlight the need for refined scaling laws that explicitly incorporate sequence length, enabling more principled choices of training configurations and potentially more efficient training strategies for future large-context models.

II. METHODS

A. Dataset

To train our models, we use text from SimpleStories – a synthetic dataset of children’s stories in very simple English. The simplicity of the language makes it possible to train small language models without them being severely undertrained.

Given that our goal is to vary context length while keeping the total number of tokens constant, we create the following datasets with varying context lengths.

a) *Chunked512*: First, we tokenize SimpleStories at the character level. This reduces the computational cost of the embedding layer while increasing the relative cost of self-attention; overall, it shifts compute toward attention, which makes our experiments feasible despite the short story lengths in this dataset. Next, we split each story into non-overlapping chunks of 512 tokens (i.e., 512 characters). We discard any final remainder shorter than 512 tokens rather than padding. As a result, we exclude approximately 2.9% of stories that are shorter than 512 characters. Nearly 60% of stories contain at least 1024 characters and therefore contribute more than one

chunk. We preserve the original SimpleStories train/validation split.

b) *Subchunked Variants*: We derive *Subchunked256x2*, *Subchunked128x4*, *Subchunked64x8*, by subdividing each chunk from *Chunked512* into smaller, non-overlapping chunks. The total number of tokens in each subchunked dataset is the same as in *Chunked512*. We preserve the train/validation split of *Chunked512* in the subchunked dataset.

B. Model Architecture

We adopt a standard Transformer decoder architecture based on the GPT design. Our models consist of stacked decoder blocks, each containing multi-head self-attention followed by a position-wise feed-forward network, with layer normalization and residual connections applied at each sub-layer.

a) *Positional Embeddings*: To enable this generalization to a longer context than a model is trained on, we employ Attention with Linear Biases (ALiBi) positional embeddings[4]. Unlike learned absolute positional embeddings or sinusoidal encodings, ALiBi applies a position-dependent bias directly to the attention scores, allowing the model to extrapolate to sequence lengths beyond those seen during training. Specifically, ALiBi adds a constant bias to each attention score based on the distance between query and key positions, with the bias magnitude scaled by a head-specific slope. This approach has been shown to provide strong length extrapolation capabilities while maintaining competitive performance on in-distribution sequence lengths.

b) *Model Scaling*:

We systematically vary model capacity across experiments to study parameter scaling laws. Model size is controlled through three architectural hyperparameters: the number of layers, the hidden dimension (model dimension), and the number of attention heads. Table I presents the three model configurations used in our experiments, spanning from 0.82M to 10.72M parameters. These model configurations are adopted from [5].

Several architectural choices remain fixed across all model configurations to isolate the effects of parameter count and context length. The vocabulary size is fixed at 88 tokens, corresponding to our character-level tokenization scheme. We use a standard feed-forward width multiplier of 4 $\times$, meaning the intermediate dimension of each feed-forward network is four times the model dimension. For the attention mechanism, we maintain a consistent head dimension by setting $d_{\text{head}} = \frac{d_{\text{model}}}{n_{\text{heads}}}$.

TABLE I
THE SHAPE OF OUR MODELS

Name	Parameters	Hidden Dim.	Layers	Attn. Heads
<i>1M</i>	0.82M	128	4	4
<i>5M</i>	4.78M	256	6	4
<i>11M</i>	10.72M	384	6	6

C. Training

We train all models using a consistent optimization setup to ensure that differences in performance can be attributed to architectural choices (parameter count and context length) rather than to variations in training procedure.

We train our models by optimizing their parameters to minimize the cross-entropy loss between the predicted logits and the true tokens. We employ the AdamW optimizer with a fixed learning rate of 1×10^{-4} . To stabilize early training, we apply a linear warmup over the first 100 optimization steps, gradually increasing the learning rate from zero to the target value. Afterwards, a cosine scheduler is used to decrease the learning rate to 0 by the end of the training. Weight decay is set to 0.01 to provide mild regularization without overly constraining the model’s capacity to fit the training data.

D. Evaluation Protocol

We compare models using the cross-entropy loss. Since modeling text with less context is inherently a harder task, the validation loss on datasets corresponding to different context lengths is not directly comparable. Additionally, in practice, we are primarily interested in models that can handle reasonably long sequences rather than models that excel only at very short contexts. Therefore, we evaluate all models using the *Chunked512* validation dataset, regardless of the context length used during training. This evaluation requires models trained on shorter sequences to generalize to longer sequences at test time; We use ALiBi positional embeddings to enable this generalization (see Section II.B.a).

E. Compute Estimation

To estimate the **floating-point operations (FLOPs)** required to train a transformer of a given architecture, we follow the methodology of [2]. In contrast to [1], our calculation explicitly includes the cost of computing token embeddings. We break down the forward-pass compute into the following components:

- **Embeddings** $2 \times \text{seq_len} \times \text{vocab_size} \times d_{\text{model}}$
- **Attention (Single Layer)**
 - Key, query and value projections: $2 \times 3 \times \text{seq_len} \times d_{\text{model}} \times (\text{head_dim} \times \text{num_heads})$
 - Key @ Query logits: $2 \times \text{seq_len} \times \text{seq_len} \times (\text{head_dim} \times \text{num_heads})$
 - Softmax: $3 \times \text{num_heads} \times \text{seq_len} \times \text{seq_len}$
 - Softmax @ query reductions: $2 \times \text{seq_len} \times \text{seq_len} \times (\text{head_dim} \times \text{num_heads})$
 - Final Linear: $2 \times \text{seq_len} \times (\text{head_dim} \times \text{num_heads}) \times d_{\text{model}}$
- **Dense Block (Single Layer)**
 - $2 \times \text{seq_len} \times (d_{\text{model}} \times \text{ffw_size} + d_{\text{model}} \times \text{ffw_size})$
- **Final Logits**
 - $2 \times \text{seq_len} \times d_{\text{model}} \times \text{vocab_size}$

For the total **FLOPs** of each forward pass, we use $\text{embeddings} + \text{num_layers} \times (\text{total_attention} + \text{dense_block}) + \text{logits}$

F. Curve Fitting

Previous works on scaling laws model the validation loss of a language model as a function of its parameter count N and the number of tokens D seen during training, as seen in (1).

Each trained model provides a data point (N, D, L) , consisting of the parameter count N , the number of tokens processed during training D , and the measured validation loss L . We fit the scaling-law model independently for each context-length run.

To estimate the five parameters of the model, we perform leave-one-out cross-validation (LOOCV). For each left-out sample i , we fit the curve on all remaining data points

$$\{(N_j, D_j, L_j)\}_{\{j \neq i\}} \quad (2)$$

and evaluate the resulting model on the held-out point (N_i, D_i, L_i) .

The parameters are optimized by minimizing the Huber loss applied in log-space:

$$\mathcal{L}_{\text{Huber}} = \text{Huber}(\log(\mathcal{L}(N, D)), \log(L))$$

For each fold, we train a fresh model using the AdamW optimizer with a learning rate of 10^{-2} , cosine annealing over 10,000 epochs, and weight decay 10^{-9} . Although we do not report formal confidence intervals due to the limited number of data points (15 data points for each context length), we assess the robustness of our scaling-law fits using leave-one-out cross-validation (LOOCV). Across all folds, the largest relative error in predicting validation loss for the held-out sample was 8%, and for the vast majority of models, we can predict their validation loss within 1% of its true value.

III. EXPERIMENTS

To systematically study how context length affects scaling behavior, we run a full grid of training configurations that vary along three independent axes: model size, training-token budget, and context length. Concretely, we train models of three sizes—*1M*, *5M*, and *11M* parameters—each on five dataset sizes corresponding to *Very Low*, *Low*, *Medium*, *High*, and *Very High* amounts of training data. These dataset sizes are defined relative to model capacity so that all models experience undertrained, moderately trained, and overtrained regimes. Finally, for every (model size, data regime) pair, we train three independent models with context lengths 128, 256, and 512 tokens. Altogether, this yields 45 distinct training runs, as seen in Fig. 1.

TABLE II
NUMBER OF TOKENS USED (MILLIONS)

	<i>Very Low</i>	<i>Low</i>	<i>Medium</i>	<i>High</i>	<i>Very High</i>
<i>1M</i>	27	54	108	216	432
<i>5M</i>	120	240	480	960	1 920
<i>11M</i>	240	480	960	1 920	3 840

The batch size is adjusted so that each training step contains the same total number of tokens, see Table III. Consequently, when two runs are trained for the same number of iterations,

they have seen identical amounts of data, differing only in how that data is arranged into longer or shorter sequences. Due to the quadratic complexity of self-attention, training with longer sequences leads to significantly higher FLOPs per iteration.

TABLE III
BATCH SIZE FOR EACH CONTEXT

Context length	Batch size	Tokens per batch
128	256	32 768
256	128	32 768
512	64	32 768

All experiments are performed on NVIDIA A100-SXM4-80GB GPUs. The computational cost varies substantially across runs, ranging from only 2 minutes for the *1M* model trained on the *Very Low* amount of data to approximately 6 hours for the *11M* model trained on the *Very High* amount of data. To ensure full reproducibility, all software dependencies are pinned and managed using Pixi, and the complete experimental codebase is [publicly available](#). This includes the data preprocessing pipeline, model configurations, training scripts, evaluation utilities, and curve-fitting code.

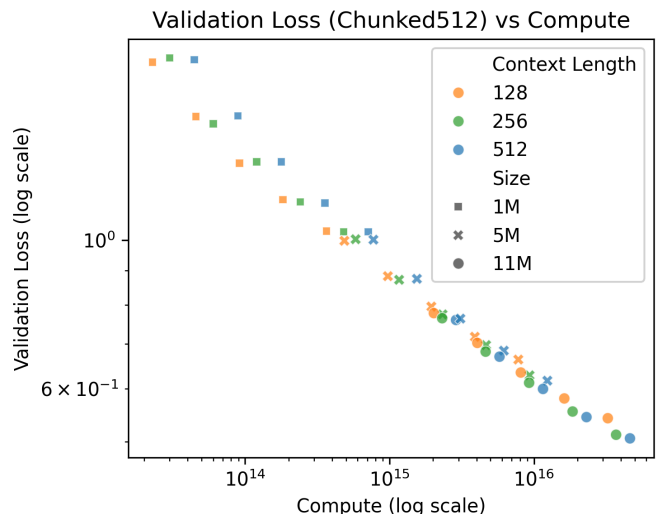


Fig. 1. Relationship between the achieved loss and the required computation. The difference in parameters for the various context lengths is small, but consistent with expectations.

TABLE IV
PARAMETERS OF SCALING LAW

Experiment	α	β	A	B	E
Context Length 128	0.34	0.33	64	250	0.92
Context Length 256	0.37	0.35	89	380	0.91
Context Length 512	0.38	0.35	105	450	0.95
[2]	0.46-0.50	0.50-0.54	410	410	n/a
[1]	0.73	0.27	n/a	n/a	n/a

A. Training Dynamics

We highlight how the validation loss decreases during the training of the *5M* model trained on the *Very High* amount of data for two context lengths: 128 and 512. When validation loss is plotted against training iterations, the longer-context run performs as well or better from the start, because it performs more computation per iteration. But when we instead plot validation loss as a function of computation performed, we see that the short context training performs better at the beginning of the training. Only later in training—once both models have processed sufficiently many tokens—does the long-context model begin to overtake, benefiting from the richer dependencies available in longer sequences. This pattern also appears when plotting validation loss against wall-clock time, though the advantage of short-context training is dampened because of fixed-time operations.

IV. CONCLUSION

As seen in Table IV, we find that the context length used during training affects the scaling law parameters. This difference is largely consistent with our expectations: a higher context length leads to larger exponents α and β (i.e., a faster decrease in loss), but also to larger A and B (i.e., higher loss in the early phases).

The E parameter captures the irreducible entropy of text—the loss that a theoretically perfect language model would achieve on our validation dataset. We would expect E to decrease with larger context length, as additional context should make each token easier to predict. It remains unclear why we do not observe this effect.

We show that, for undertrained models, it can be beneficial to train with shorter context lengths. This effect is visible in Fig. 1, where training runs corresponding to the three context lengths follow three distinct trends. Shorter context length has a clear advantage when less than 10^{15} FLOPs is used.

We further illustrate this phenomenon in Fig. 2. Two *5M* models trained on a *Very High* amount of data are trained using sequences of length 128 and 512, respectively. When plotting performance (validation loss) against iterations, we observe that training on longer sequences leads to greater improvements only later in the process. However, since training on longer sequences requires more FLOPs, training on shorter sequences is more efficient in the early phase.

We stress that this effect is present despite all models being evaluated on 512-long sequences. In the undertrained regime, models trained on 128-long sequences outperform the model trained on 512-long sequences, despite never seeing sequences of that length during training. This is a testament to the effectiveness of the Alibi embeddings [4].

However, the overall scale of our experiments remains small. Moreover, the Simple Stories dataset consists of narratives that are too short to fully stress-test context-length generalization. While character-level tokenization allowed us to artificially lengthen inputs, this setup is ultimately unrealistic and limits the generality of our conclusions.

Future work should attempt to reproduce our findings on a more realistic training regime: larger models trained on

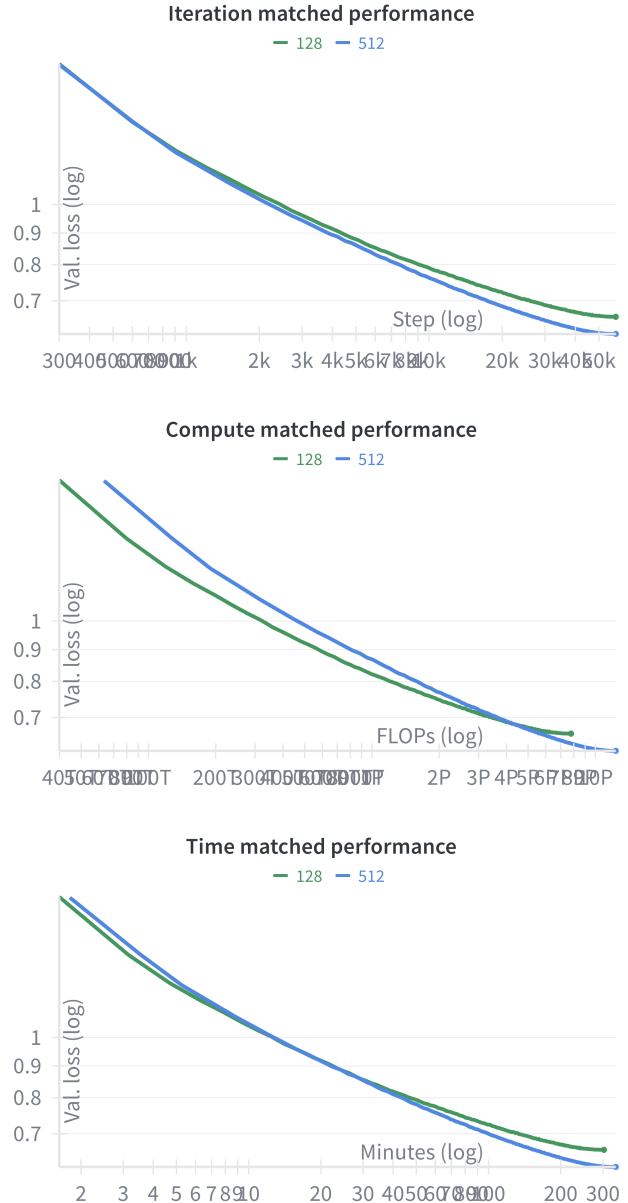


Fig. 2. Two *5M* models trained on the *Very High* amount of data, using 128 and 512 long sequences respectively. Both are evaluated on 512 long sequences.

longer sequences. Furthermore, they should provide principled guidance on selecting and scheduling context length during training. In particular, understanding how to dynamically adjust context length—and quantifying the resulting training speedups—represents an important direction. We hope that our findings motivate more systematic exploration of context-length allocation as a resource to optimize in the training of large language models.

REFERENCES

- [1] J. Kaplan *et al.*, “Scaling Laws for Neural Language Models.” Accessed: Sept. 01, 2025. [Online]. Available: <http://arxiv.org/abs/2001.08361>

- [2] J. Hoffmann *et al.*, “Training Compute-Optimal Large Language Models.” Accessed: Sept. 21, 2025. [Online]. Available: <http://arxiv.org/abs/2203.15556>
- [3] Q. Yang, Q. Peng, H. Liu, K. Liu, B. Qin, and T. Liu, “Beyond Fixed Length: Bucket Pre-training is All You Need.” Accessed: Dec. 13, 2025. [Online]. Available: <https://arxiv.org/abs/2407.07495v2>
- [4] O. Press, N. A. Smith, and M. Lewis, “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation.” Accessed: Oct. 14, 2025. [Online]. Available: <http://arxiv.org/abs/2108.12409>
- [5] L. Finke *et al.*, “Parameterized Synthetic Text Generation with SimpleStories.” Accessed: Sept. 22, 2025. [Online]. Available: <http://arxiv.org/abs/2504.09184>